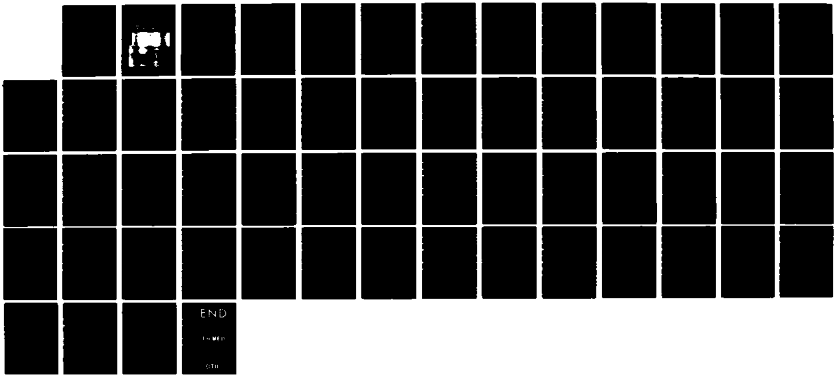
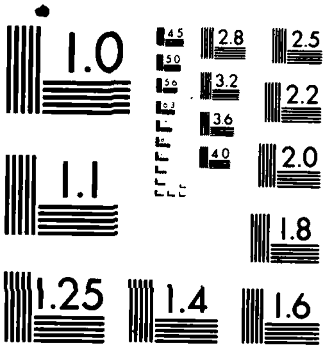
AD-A161 552   A PERFORMANCE ANALYSIS OF MULTIPROCESSORS USING   1/1
TWO-LEVEL CACHES(U) ILLINOIS UNIV AT URBANA COMPUTER
SYSTEMS GROUP D J COLGLAZIER AUG 84 CSG-36
UNCLASSIFIED   N00014-84-C-0149   F/G 9/2   NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

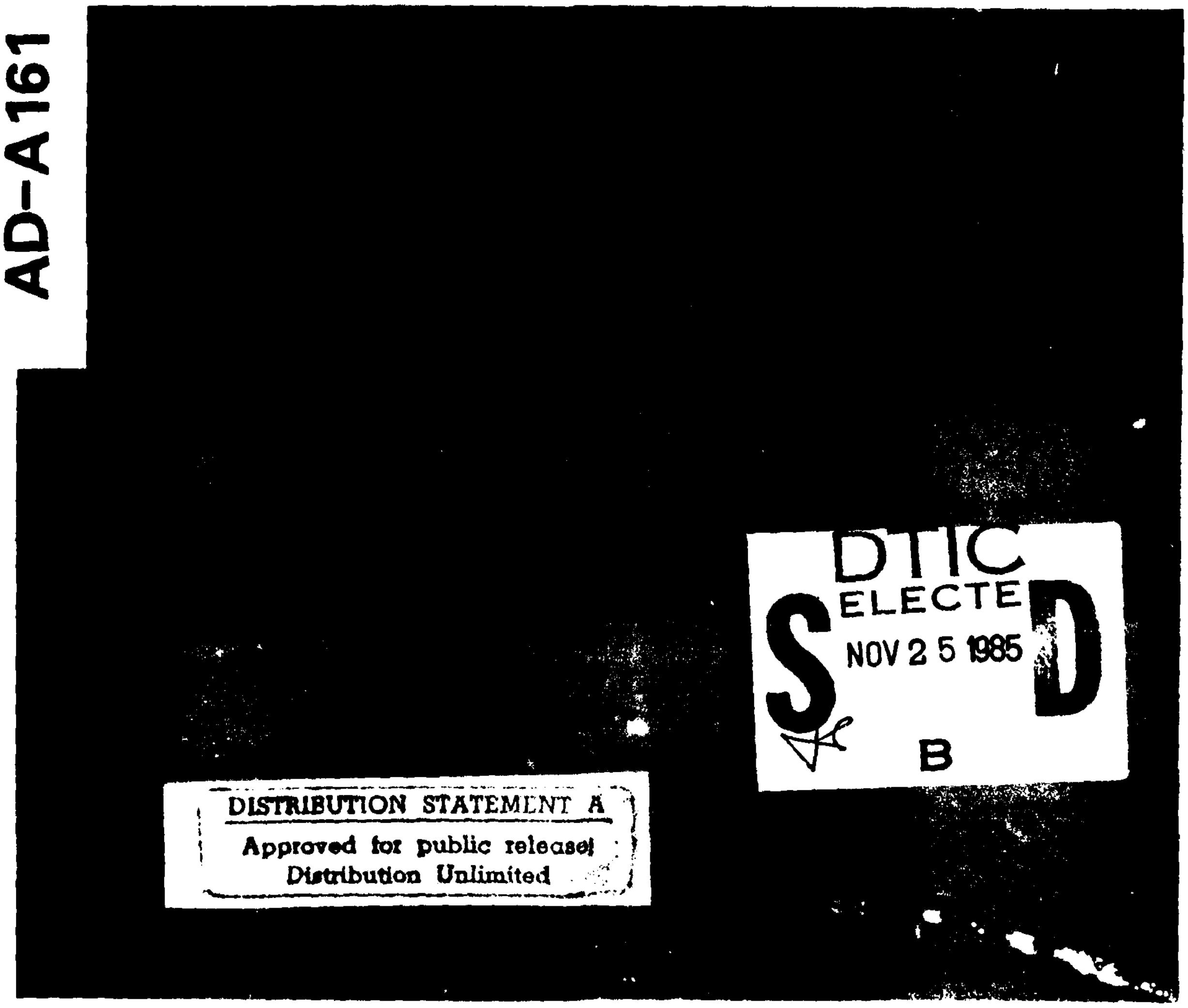
12

REPORT CSG-36 AUGUST, 1984

COORDINATED SCIENCE LABORATORY

COMPUTER SYSTEMS GROUP

A PERFORMANCE ANALYSIS OF MULTIPROCESSORS USING TWO-LEVEL CACHES



UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A161 556

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
N/A

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
R-1020 (CSG-36); UILU-ENG 84-2214

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Coordinated Science Laboratory
University of Illinois

6b. OFFICE SYMBOL *(If applicable)*
N/A

7a. NAME OF MONITORING ORGANIZATION
Joint Services Electronics Program

6c. ADDRESS *(City, State and ZIP Code)*
1101 West Springfield Avenue
Urbana, IL 61801

7b. ADDRESS *(City, State and ZIP Code)*
Office of Naval Research
800 North Quincy Street
Arlington, VA

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Joint Services Electronics Program

8b. OFFICE SYMBOL *(If applicable)*
N/A

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
N00014-84-C-0149

8c. ADDRESS *(City, State and ZIP Code)*
Office of Naval Research
800 North Quincy Street
Arlington, VA

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
N/A	N/A	N/A	N/A

11. TITLE *(Include Security Classification)* A Performance Analysis of Multiprocessors Using Two-Level Caches

12. PERSONAL AUTHOR(S)
Colglazier, Daniel James

13a. TYPE OF REPORT
Technical

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Yr., Mo., Day)*
August 1984

15. PAGE COUNT
55

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
multiprocessors, cache memories, two-level cache memories, cache coherence, performance analysis, multiprocessor memory

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report proposes a two-level cache organization for multiprocessors. The first level of cache consists of a private cache per processor. The second level of cache is shared by all processors. The main memory is also similarly shared. A cache coherence solution is proposed for such an organization. The performance of the proposed multiprocessor is evaluated with analytical methods. The factors that affect the performance are quantitatively discussed. A variation of the proposed coherence scheme is presented to improve the performance.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL

22b. TELEPHONE NUMBER *(Include Area Code)*

22c. OFFICE SYMBOL
N/A

EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

A PERFORMANCE ANALYSIS OF MULTIPROCESSORS USING TWO-LEVEL CACHES

BY

DANIEL JAMES COLGLAZIER

B.S., University of Illinois, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1984

Urbana, Illinois

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Multiprocessor computer systems offer many advantages over single processor computers. The multiprocessor can be devoted to a single job reducing its overall execution time. This also allows more difficult jobs to be attempted. Multiple jobs can also be run simultaneously increasing the throughput of the system. Shared resources are efficiently utilized because the average workload will contain a mix of processor-bound and memory-bound jobs. Multiprocessors are very reliable. They can be designed so that the failure of one processor only degrades the system perfomance, not crashes the entire system. A multiprocessor can also be designed so that processors can be added as needed. Multiprocessors can therefore be tailored for specific applications or grow incrementally to meet demand. A tightly coupled multiprocessor is a computer system containing two or more processors that can execute instructions independently, that are not highly specialized, and that share most or all of memory. Many multiprocessor computer systems do not share memory, but these will not be considered in this thesis. Therefore, the term multiprocessor will be used to mean tightly coupled multiprocessor hereafter.

As the number of processors in a multiprocessor increases, the throughput of each processor decreases. This is due mainly to memory interference. The effects of memory interference can be reduced by increasing the number of parallel memory modules. Therefore, a multiprocessor system requires a large number of memory modules and as a result a large processor to memory switch. Using faster processors is another way to increase the throughput. In order to efficiently utilize fast processors, memory access time must be close to the cycle time of the processors. Memory access time increases with the size of the switch. However, fast processors require fast access time. The solution to achieve a fast access time with a large shared memory is to use a memory hierarchy. A *memory hierarchy* is a storage system of two or more levels of storage with dissimilar technologies, capacities, access times, and per-unit costs. Using a program's property of locality, memory accesses are often completed at the speed of the fastest and smallest memories while the cost per unit of storage approaches that of the least costly and slowest storage. The need to keep the cost of memory low is another motivation for memory hierarchies.

Most recently produced computers have cache memories as the highest level of their memory hierarchy. Examples include large computer systems such as the IBM 3081 as well as medium and small computers such as the DEC VAX 11/780 and PDP-11/70. Cache memories are designed to hold temporarily those memory words believed to be currently in use. When a processor makes a memory request, the cache is checked to see if it holds the desired word. If the word is in the cache, a hit occurs and the word is sent to the processor at a speed close to the processor cycle time without accessing main memory. If the word is not in the cache, a miss occurs. A miss is processed by bringing into the cache the block in main memory containing the word and sending it to the processor. Room must be found in the cache for this new block which typically involves replacing a previously cached block. Blocks are usually larger than the largest unit of data that the processor can access in order to exploit the program's property of locality. It is not the intention of this thesis to document the operations and advantages of cache memory, so an acquaintance with cache principles will be assumed. Refer to [Conti69], [Meade70], [Kaplan73], [Strecker76], [Rao78], [Smith82] for additional background on cache memories.

The cache *miss ratio* is the fraction of processor requests not found in the cache. It is easy to see that a low miss ratio is needed for good performance. Usually, as the size of the cache increases, its miss ratio decreases. Therefore, the cache must be large enough to provide a low miss ratio. Unfortunately, as the cache size grows, its access time increases which defeats its purpose. It also becomes more expensive. A solution to this problem is to extend the memory hierarchy concept to the cache memory creating a two-level cache. This thesis studies the performance of two-level cache configurations in multiprocessor computer systems.

Each cache level can either be shared by all processors or split into a number of independent caches, one for each processor. This latter configuration will be referred to as private cache. Shared cache offers many advantages over private cache [Yeh83]. Private cache is constrained to a fixed cache allocation per processor while shared cache allows for the dynamic allocation of the total cache space. Memory words requested by more than one processor have to be brought into each private cache while only one copy is needed in a shared cache. For these reasons, a shared cache can attain a lower miss ratio. Interprocessor communication can easily be achieved through a shared cache. Shared cache also avoids the cache coherence problem. This problem occurs in a private cache configuration when more than one private cache holds the same memory block and one processor modifies its copy. Unfortunately, a shared cache causes performance degradation due to access conflicts. In order to obtain reasonable performance, the number of cache modules should be greater than the number of processors [Yeh83]. Since one of the reasons for using a two-level cache is to efficiently utilize fast processors, the first level of the cache (L1 cache) will be composed of private caches. Most processor requests will be found in L1 cache, so the second level of the cache (L2 cache) will receive far fewer requests than L1 cache. Therefore, the access conflict problem is not as serious in L2 cache allowing it to be a shared cache. Figure 1-1 shows the proposed two-level cache configuration.

In Chapter 2, the cache coherence problem is examined. Section 2.1 defines cache coherency and describes the sources of the coherence problems. It also reviews some general techniques for dealing with these problems. Specific cache coherence algorithms are

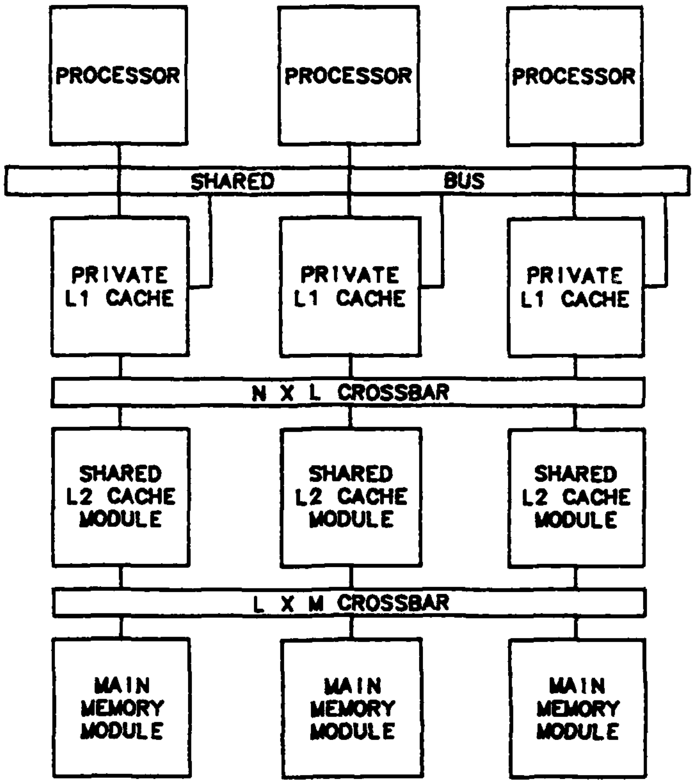


Figure 1-1. Proposed two-level cache configuration.

presented in the next two sections. Section 2.2 describes a number of proposed solutions that utilize directory information kept in main memory to insure cache coherence. These are called directory schemes. In Section 2.3, various proposed bus schemes are presented. These algorithms distribute the directory information among the caches and use a bus to broadcast needed information. A coherence solution for the two-level cache is proposed in Section 2.4. Ideas from the previously proposed cache coherence solutions are adopted into an algorithm appropriate for the proposed two-level cache configuration.

In Chapter 3, the performance of the proposed two-level cache configuration and coherence solution is evaluated. Section 3.1 presents a performance model for the proposed two-level cache and the jobs run on it. An approximate analytical method is used to derive a number of equations in Section 3.2. These equations can be solved by numerical analysis to evaluate the performance of the two-level cache. The results of the performance analysis are discussed in Section 3.3.

It will be shown in Chapter 3 that the bus needed for cache coherence is the bottleneck of the system. Chapter 4 presents a method that attempts to reduce the bus usage. The previously derived equations are modified to evaluate the performance of this bus usage reduction method. The results of the performance analysis are presented at the end of Chapter 4.

In the final chapter, the future applicability of multiprocessors with two-level caches is discussed. Areas where further research is needed are also pointed out.

CHAPTER 2

CACHE COHERENCE ALGORITHMS

2.1. The Cache Coherence Problem

A multiprocessor system is cache coherent if a read request for any word always returns the most recently written value of that word. The private cache configuration chosen for L1 cache has coherence problems, as previously mentioned. The problems come about when memory blocks exist in more than one private cache. Multiple copies of memory blocks in private caches is the result of more than one job requesting the same memory block or a job being switched from one processor to another (task switching) that requests the memory block while in each processor. If one copy of a memory block is modified, all other copies must be informed of this change to guarantee that all subsequent read requests to that block from any processor return this updated version.

If no shared and writable memory blocks are allowed to be cached, then no cache coherence problems exist. Since task switching makes any memory block potentially sharable, the above solution limits cached blocks to be read-only. This can be implemented by requiring the operating system to designate which memory blocks can be cached. A special operating system is then required since cache memories are usually designed to be transparent to the operating system. Poor performance results when jobs with high write rates are executed. Much better performance can be achieved by allowing writable memory blocks to be cached.

The classical coherence solution uses a data path connected to every cache over which all the processors send the addresses of the blocks that they modify. Each cache monitors the data path and invalidates or updates those blocks that it contains. In addition, the modification is sent to main memory (write-through policy) so that it always contains the most recently written value of the block. The main drawback of this scheme is the high traffic on the data path. As the number of processors in the system increases, this traffic increases to the point that the data path becomes the bottleneck of the system. Also, every cache in the system must surrender a cycle to check to see if it has the block that has been modified. This causes much cache interference. The problem can be reduced by using a bias filter [Bean79] which is a small memory associated with each cache. It keeps track of the most recently invalidated blocks so that repeated invalidations to the same block only interrupt a cache once.

The best place to reduce data path traffic and cache interference is at the source, the caches themselves. Two classes of coherence solutions exist that attempt to do this. One scheme is to use a directory to keep track of which blocks are currently being shared. Only write requests to these shared blocks need to be sent on the data path. The other scheme is to connect all the caches to a bus and broadcast write requests on the bus. The directory information in the previous scheme is distributed among the caches to reduce bus traffic. Specific proposals of directory schemes are presented in the next section and specific bus schemes are reviewed in the section following that.

2.2. Directory Schemes

The central directory scheme [Tang76] uses a "store controller" to control the communication among the caches. Cached blocks are declared shared or private. Shared blocks may exist in more than one cache and are consistent with their corresponding main memory block. Only one private block can exist among all the caches. A private block is one that has been modified or has a write request to it pending. The store controller must know at all times the status of every cache block. The central directory is used for this purpose. It contains a duplicate of each cache's directory. For a processor to modify a previously unmodified block, it must first issue a request to the store controller to convert the shared block to a private block. If the "shared" block was actually shared by more than one cache, these additional copies are invalidated. The store controller can also cause a cache to send a private block to main memory and then change its status to shared, or invalidate it. This scheme requires extensive central directory searching for each cache miss and block status checking for each cache write. This scheme works well for a small number of processors, but access conflict problems occur at the central directory for a larger number of processors. The central directory also becomes prohibitively large for a large number of processors.

A similar coherence solution is the presence flag solution [Censier78]. Its main objective is to eliminate all or almost all ineffective invalidation requests. Each block in cache contains a private flag to indicate whether the block is private or shared. Each main memory block contains a presence flag for each cache in the system. The presence flag indicates whether or not the memory block is in a particular cache. Each main memory block also contains a modified flag to indicate whether or not the block has been modified in a cache. The presence flags allow the invalidation and update requests to be sent only to those caches whose presence flag is set. The S-1 Multiprocessor uses a coherence solution similar to this [Widdoes79]. The main drawback of this scheme is the extra memory required for the presence flags.

The objective of the logical semi-critical section scheme [Yen82] is to fulfill as many requests as possible entirely in a private cache. In the previous two proposals, a processor can not commence writing into an unmodified block until the state of the corresponding main memory block is changed even if its cache contains the only copy. This scheme allows write requests to be executed on unmodified and unshared blocks without first consulting a global directory. The same global information used in the presence flag solution is used, but more local information is kept for each cache block. Each cache block contains a valid bit, a single bit, and a modify bit. Different combinations of these bits produce four possible states for each cache block: invalid, single-unmodified, shared-unmodified, and single-modified. Main memory and each cache has a separate controller to keep track of all the status information. The key to this scheme is the local single states. These states are maintained by checking if any other cache holds a copy of the block when it is first read in, and checking if a single copy remains when a shared block is replaced. This scheme provides a performance improvement over the previous presence flag scheme, but requires even more memory overhead.

The amount of extra memory needed can be greatly reduced by using the two-bit solution [Archibald84]. Each main memory block is in one of four states: absent, private-unmodified, shared-unmodified, or private-modified. These states are encoded using only two bits and describe the state of each main memory block in the caches. For example, the private-unmodified state means this memory block is in only one cache and has not been modified. The smaller main memory block tag does not allow the identities of the caches holding a particular block to be known. Invalidations and status updates must be broadcast to all the caches. This scheme is therefore not as time efficient as the previous ones, but is much more memory space efficient. Another advantage is that it is easily expandable as its main memory block tags are the same size for any number of processors.

2.3. Bus Schemes

Bus schemes differ from directory schemes in that no tag bits are kept in main memory. The directory scheme's global information is distributed among the individual caches. A shared bus is used to broadcast requests among the caches. The write-once scheme [Goodman83] is designed to produce minimal bus traffic. Each block in a cache contains two bits indicating that it is in one of four states: invalid, valid, reserved, or dirty.

The valid state indicates that the block is unmodified. The reserved state means that the block has been modified once and its corresponding main memory block has been updated. The dirty state signifies that the block has been modified more than once and its corresponding main memory block has not been updated. The first write to an unmodified (valid) block causes the block's status to be changed to reserved, its corresponding main memory block to be updated, and any other copies of this block in other caches to be invalidated. A write to a reserved block changes its status to dirty. No other cache is informed of this write as it is guaranteed that no other cache holds a reserved or a dirty block. When a dirty block is replaced in a cache, it must be written back to main memory. No other block type is required to do this. Therefore, this scheme uses a write-through policy for the first write to a block and a write-back policy for subsequent writes to that block.

Another important feature of this scheme is that it uses dual directories. One directory is used conventionally to help process requests made by its processor. The second directory monitors the bus for requests for blocks contained in that cache. Both directories hold the same address data and are always updated simultaneously. The second directory reduces cache interference because each cache is interrupted only when the bus request is for a block it currently holds. The write-once scheme is similar to the classical solution, but attempts to reduce both of its drawbacks: high traffic intensity and large cache interference.

The RWB scheme [Rudolph84] is in many ways an extension of the write-once scheme. The main difference is in how a broadcasted write request is handled. In the write-once scheme, when a cache receives a write request over the bus to a block it holds, the block is invalidated. In the RWB scheme, the written data is broadcast on the first write to an unmodified block and all caches holding that block are updated. Subsequent writes to that block are treated as in the write-once scheme. This scheme produces a lower miss ratio since fewer blocks will be invalidated, but increases the bus usage. If a block is written to more than once, the write-once scheme broadcasts over the bus just once while the RWB scheme broadcasts over the bus twice. To deal with the higher bus traffic, the use of multiple shared buses is suggested. This involves splitting the caches using the least significant address bits and using each part of the divided cache to monitor a separate bus.

In [Papamarcos84], a low-overhead coherence solution is presented and its performance is analyzed. This solution is also similar to the write-once scheme, but a write-back policy is used for all write requests. Two status bits are associated with each block in a cache: a shared/exclusive bit and a modified bit. All combinations of these bits are allowed except for the shared and modified state which is used for the encoding of the invalid state. Dual directories are utilized, but the second directory only uses one status bit along with the address data to indicate whether the block is present or invalid. Write requests to a block cause its state to become exclusive-modified. An invalidation signal is broadcast for a write to a currently held block only if its status was shared-unmodified. Replaced cache blocks are written back to main memory only if their status was exclusive-modified. Upon a cache miss, a read request is broadcast on the bus to all caches and main memory. If a cache holds the block, it responds with the data and changes its status to shared. If the data had been modified, it is written back to main memory at the same time. The new cache block is declared shared or exclusive depending on whether or not another cache holds a valid copy of that block. The shared-unmodified state indicates only that the block may be present in more than one cache. When the block was initially declared shared-unmodified, it came from another cache guaranteeing that at least two caches held this block. When all but one of these blocks have been replaced, this one block is no longer shared. Unlike the logical semi-critical section scheme, the block's status is not changed to simplify implementation. The performance analysis showed that very little performance degradation resulted from using this coherence solution.

2.4. Two-Level Cache Coherence Solution

The goal of the two-level cache configuration is to fulfill requests from the processors at the highest possible level without accessing lower levels. Private caches have been chosen for L1 cache so that fast processors can be efficiently utilized. The private caches must be able to operate independently with as little interference as possible. These things must be kept in mind when deciding how to implement a two-level cache coherence solution. The cache coherence solutions reviewed in the previous sections are all intended for one level of private caches backed by main memory. The proposed two-level cache configuration adds a shared cache between the private caches and main memory. Main memory holds all or most of the global status information for the cached blocks in the directory schemes. If a directory scheme is used for the proposed two-level cache configuration, L2 cache will do little to reduce the number of main memory accesses from the processors. A bus scheme therefore seems more appropriate for the proposed two-level cache configuration.

A major drawback of bus schemes is the amount of interference that a "private" cache suffers due to requests broadcast by the other caches. The dual directory greatly reduces this interference and will be used for the two-level cache coherence solution. Interference still occurs in the caches for invalidations, status checks and updates, and requests for data transfers over the bus for the blocks they hold. Requests for data transfers usually are accompanied by status updates, but it seems reasonable to assume that the status updates can be processed faster than data transfers. Therefore, interference can be reduced by not allowing data transfers on the bus. The performance degradation due to this restriction should not be too severe since most blocks in L1 cache will also be in L2 cache. Block transfers from L2 cache are not much slower than from another private L1 cache and do not use the bus. This restriction requires that all modifications of blocks in L1 cache be reflected in L2 cache. Therefore, a write-through policy to L2 cache will be used. This write-through policy does not affect the bus traffic because L2 cache is not attached to the bus. Each private L1 cache will contain a buffer to store several write requests so that its processor will not have to wait for them to be completed. Very little (if any) performance degradation should result from the selection of this write policy.

The other major drawback of bus schemes is the high bus traffic. The two-level cache coherence solution will strive to minimize this traffic. Not allowing data transfers reduces the bus traffic, as previously mentioned. The number of invalidations and status checks and updates must now be minimized. Invalidations are broadcast for write requests whenever another private L1 cache possibly holds a copy of the block written into. This means that all write requests must be broadcast except to those cached blocks guaranteed not to have any other copies in L1 cache. It is advantageous, then, to know if a block is shared or exclusive. A block is declared shared or exclusive when it is brought into a private L1 cache depending on whether or not another private L1 cache already holds a copy of the block. An exclusive block remains in that state unless another private L1 cache broadcasts a read request for that block. When a read request broadcast on the bus hits an exclusive block, it is changed to a shared block. A shared block can become exclusive only if it is written into. A write to a shared block causes a write request to be broadcast on the bus invalidating all other copies of the block. Shared blocks are not updated to exclusive blocks when all but one copy of the block is replaced. This may cause unnecessary invalidates to be broadcast on the bus, but is better than the alternative. To determine when only one copy of a shared block is left in L1 cache, all shared block replacements must be broadcast over the bus to see if only one private L1 cache now holds the block. This causes at least one extra bus broadcast, while leaving the block as shared causes at most one unnecessary bus broadcast. Each block in L1 cache therefore will have two bits associated with it in each directory: one to indicate whether it is shared or exclusive and another to indicate whether it is valid or invalid.

When a processor requests to read a word, its private L1 cache is checked to see if it is present there. If it is, it is sent to the processor. The bus is not used and no status changes take place. If the word is not found in its private L1 cache, the read request is broadcast on the bus to all the private L1 caches. If the word is found in another private L1 cache, that cache responds with a hit signal on the bus and changes its block's status to shared if it had been exclusive. While this broadcast takes place, L2 cache is checked to see if it contains the block containing the word. If it does, the block is sent to the private L1 cache of the processor making the read request through the $N \times L$ crossbar switch. If the word is not found in L2 cache, the block containing the word is found in main memory, sent to L2 cache through the $L \times M$ crossbar switch, and then sent to the requesting private L1 cache through the $N \times L$ crossbar switch. The processor reads the word from its private L1 cache once it arrives there. The block brought into L1 cache is declared shared if the read broadcast returns a hit, or exclusive if no hit is returned. Figure 2-1 is a flow chart showing the required operations to fulfill a read request.

Processing a write request is a little more complicated. If the block containing the word is found in a processor's private L1 cache, the word is updated. If the word was found in a shared block, a write request must be broadcast on the bus to invalidate all other copies. No bus activity results from writing to an exclusive block. If the word is not found in a processor's private L1 cache, a write request must be broadcast on the bus since it possibly exists in another private L1 cache. It must now be decided whether or not to allocate space for the word's block in L1 cache. The processor does not need any data from the block to complete its write request, so the block need not be brought into L1 cache. Bringing in the block ties up the cache and usually replaces a block that may be needed for a future read request. An alternative is to allocate space for the block without bringing it

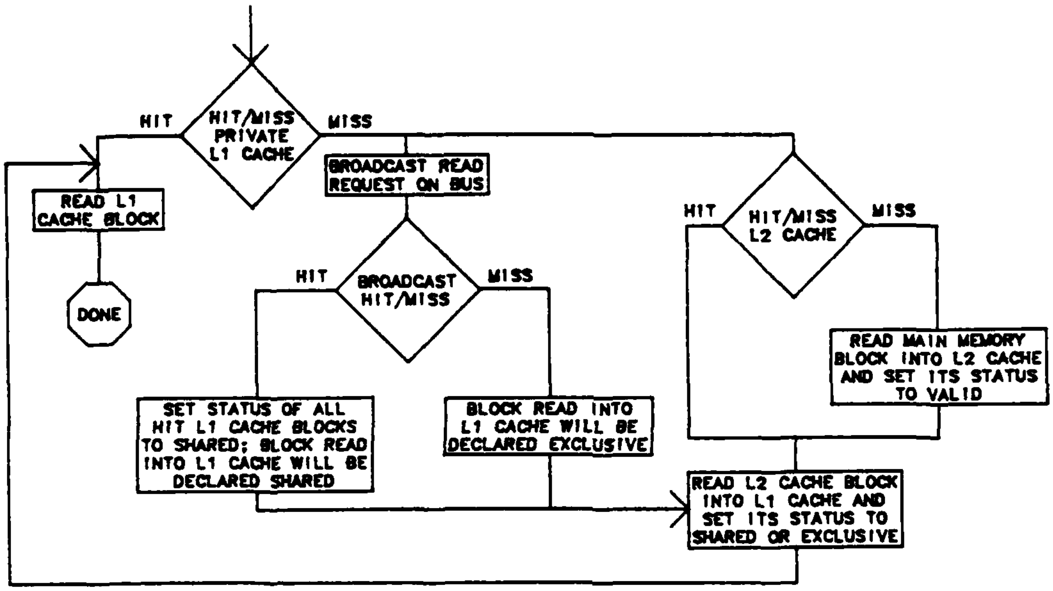

HIT
HIT/MISS PRIVATE L1 CACHE
MISS
READ L1 CACHE BLOCK
DONE
BROADCAST READ REQUEST ON BUS
HIT
BROADCAST HIT/MISS
MISS
SET STATUS OF ALL HIT L1 CACHE BLOCKS TO SHARED; BLOCK READ INTO L1 CACHE WILL BE DECLARED SHARED
BLOCK READ INTO L1 CACHE WILL BE DECLARED EXCLUSIVE
HIT
HIT/MISS L2 CACHE
MISS
READ MAIN MEMORY BLOCK INTO L2 CACHE AND SET ITS STATUS TO VALID
READ L2 CACHE BLOCK INTO L1 CACHE AND SET ITS STATUS TO SHARED OR EXCLUSIVE

Figure 2-1. Read request operations.

in and to write the word into the allocated space with the rest of the block being invalid. This a is faster solution, but increases the miss ratio because a totally valid block is usually replaced. Since a write-through policy to L2 cache is being used, a write request missing in a private L1 cache need not affect L1 cache at all. No blocks will be replaced, but bus traffic will be increased. Allocating space for the block creates an exclusive block with subsequent writes to it not being broadcast on the bus. Not affecting L1 cache causes subsequent writes to the same block to be broadcast on the bus as they will result in L1 cache misses. It is difficult to determine what the best alternative is. The no-allocate policy is chosen to keep the performance analysis from being too difficult.

Regardless of whether or not the word's block is found in a processor's private L1 cache, it is written into a buffer in the processor's L1 cache controller. This write is done simultaneously with the write into the private L1 cache if the word's block is present there. When the write to this buffer has been completed, the processor is free to continue. Figure 2-2 shows the operations performed by a processor to complete a write request.

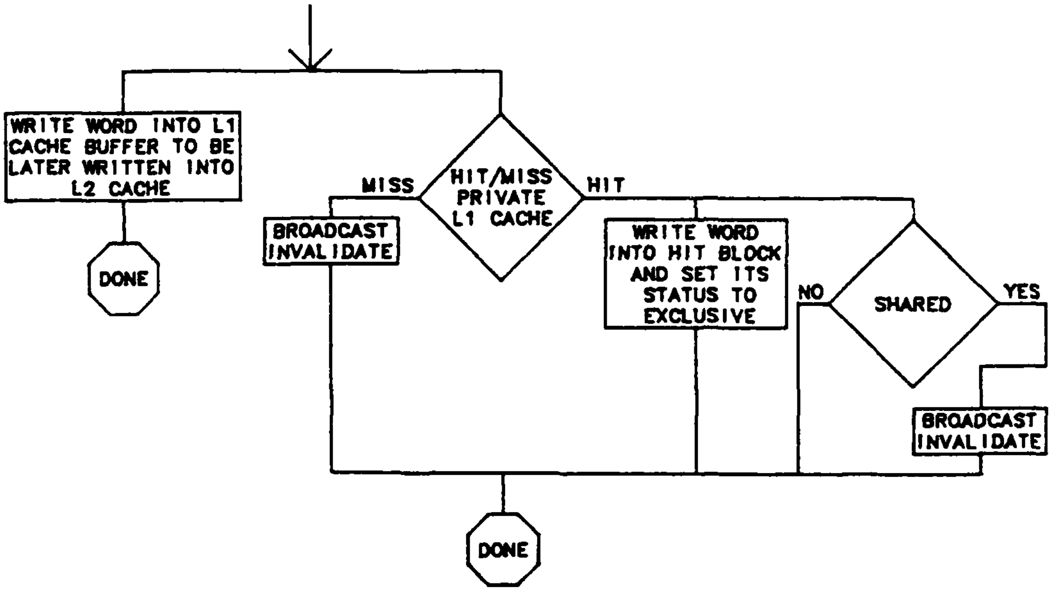

WRITE WORD INTO L1 CACHE BUFFER TO BE LATER WRITTEN INTO L2 CACHE
DONE
MISS
HIT/MISS PRIVATE L1 CACHE
HIT
BROADCAST INVALIDATE
WRITE WORD INTO HIT BLOCK AND SET ITS STATUS TO EXCLUSIVE
NO
SHARED
YES
BROADCAST INVALIDATE
DONE

Figure 2-2. Write request operations performed by a processor.

Each L1 cache controller is responsible for sending the words in its buffer to L2 cache. The L1 cache controllers must establish a path to the appropriate L2 cache module through the $N \times L$ crossbar switch and send the word along the path. When L2 cache holds the block for the word, it is simply updated. If the block is not present, the question again arises whether or not to bring the block into L2 cache. The same technique can be used at an L2 cache module as is being used at the private L1 caches. The word can be written into a buffer in the appropriate L2 cache module controller and later sent to main memory. This write-through, no-allocate technique for L2 cache quickly processes the L1 cache write-throughs. The operations of the L1 cache controllers for a write request using this write-through policy are shown in Figure 2-3. If this technique causes too much main memory traffic, a write-back technique can be used. A write to a block not found in L2 cache will cause the block to be transferred from main memory. When the block is in L2 cache, the word is written into the block. Main memory is not updated until the block is replaced in L2 cache. The write-through technique requires that L2 cache blocks be declared valid or invalid, while the write-back technique requires an extra status bit to determine if the block has been modified or not upon replacement. Only modified blocks need to be sent to main memory. The performance of both techniques will be analyzed in the next chapter.

Shared blocks allow simultaneous accesses to a block by different processors. Any instruction using a shared block that might be interfered with by another processor simultaneously accessing the block will be required to first gain control of the bus before proceding. It will invalidate the other shared blocks causing accesses to those blocks to miss. Any memory request that misses must use the bus. By requiring these requests to control the bus before completing, problems from simultaneous accesses are eliminated because only one cache can control the bus at a time. Only the memory requests from instructions are considered in the next chapter. Simultaneous memory reads are allowed, but simultaneous memuiy writes are not. A write request to a shared block must broadcast the write request

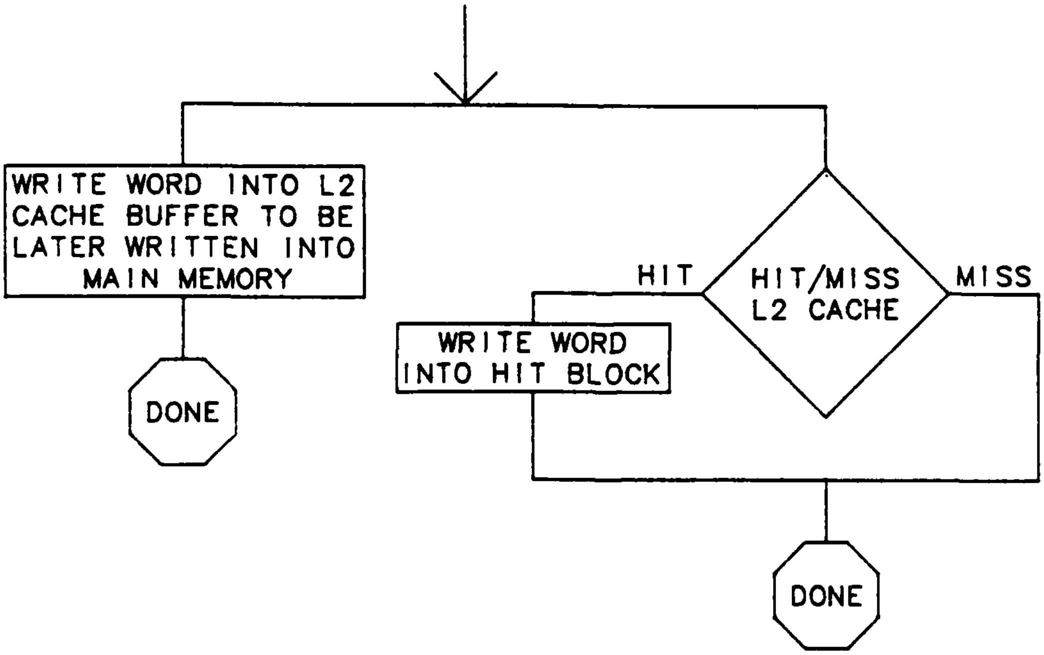

WRITE WORD INTO L2 CACHE BUFFER TO BE LATER WRITTEN INTO MAIN MEMORY
DONE
HIT
HIT/MISS L2 CACHE
MISS
WRITE WORD INTO HIT BLOCK
DONE

Figure 2-3. Write request operations performed by an L1 cache controller.

on the bus. Simultaneous write requests to the same block will not therefore cause any difficulty. The first one to capture the bus will invalidate the other block, write into its own cache, and declare it exclusive. The second one to capture the bus will find its block invalidated and proceed as a write request that misses in L1 cache. This involves a broadcasted write request which invalidates the block written in L1 cache by the first one. The only resultant valid copies of this block are therefore in L2 cache and/or main memory.

CHAPTER 3

PERFORMANCE EVALUATION

3.1. Performance Model For The Proposed Two-Level Cache

The performance of the proposed two-level cache configuration will be modeled using a number of parameters. These parameters attempt to completely specify the behavior of both the two-level cache and the jobs run on it. The parameters are defined as follows:

N	Number of processors
L	Number of L2 cache modules
M	Number of main memory modules
m_1	Private L1 cache miss ratio
M_2	Shared L2 cache miss ratio
T_{12}	Number of cycles to transfer a block from L2 cache to L1 cache
T_{2mm}	Number of cycles to transfer a block from main memory to L2 cache
T_{s112}	Number of cycles to write one word into an L2 cache buffer by an L1 cache controller
T_{s1mm}	Number of cycles to write one word into main memory by an L2 cache controller
a	Percentage of processor cycles that access memory
w	Percentage of memory accesses that are write requests
s	Percentage of memory accesses that reference blocks that are declared shared
I	Number of cycles the bus is occupied for an invalidation broadcast
A	Number of cycles needed for bus arbitration logic

Each processor has its own cache, so N is also the number of private L1 caches. Each of the L shared L2 cache modules contains part of the L2 cache memory and its own controller and functions independently. The M main memory modules also function independently. The remaining parameters require a more thorough explanation.

m_1: The miss ratio of each private L1 cache is affected by many things: the cache size, the block size, when the blocks are brought in, how the blocks are placed and replaced, the frequency of task switching, and program behavior. These effects are not considered individually, but are all considered to be part of m_1. A single miss ratio is used for all memory accesses, both read and write requests. It is assumed that either the difference for the two requests is small or that an average of the two is used.

M_2: The parameter M_2 is the miss ratio of the shared L2 cache calculated as if the L1 and L2 caches are combined. Therefore, it is the miss ratio of all memory accesses. The explanation given for m_1 also applies to M_2. A more useful parameter for measuring performance is the miss ratio of L2 cache for only those memory accesses not found in L1 cache. This miss ratio will be defined as m_2. L2 cache should be large enough to hold most of the blocks in the private L1 caches. The parameter M_2 is then the product of m_1 and m_2 as it is the percentage of memory accesses not found in a private L1 cache nor found in L2 cache. The parameter m_2 is therefore simply: $\frac{M_2}{m_1}$.

T_{12}: The block transfer time from L2 cache to L1 cache through the $N \times L$ crossbar depends on the bandwidth of the path and the size of the block transferred. The transfer time does not include the wait time if the path or cache module is already being used. These wait times will be considered separately.

T_{2mm}: This parameter is similar to T_{12}, but applies to transfers through the $L \times M$ crossbar between L2 cache and main memory.

T_{s12}: All write requests are written into a buffer in the private L1 caches so that the processors do not have to wait for the word to be written through to L2 cache. The L1 cache controllers take the words from their buffers and write them to the appropriate L2 cache module's buffer as well as to L2 cache if its block is present there. This write time also does not include the wait time if the L2 cache module is already busy, a quantity that will be considered separately.

T_{s1mm}: This parameter is similar to T_{s112} except that the words are written out of the L2 cache buffers through the $L \times M$ crossbar to the appropriate main memory block.

a: Only instructions that access memory are considered in this model. The parameter a is the fraction of processor cycles that access memory. This parameter combines the effects due to the computer's instruction set and use of registers as well as program behavior. Some programs are processor-bound while others are memory-bound.

w: Memory accesses are either write requests or read requests. The parameter w is the fraction of memory accesses that are write requests. The remaining $1-w$ memory accesses are therefore read requests.

s: All valid L1 cache blocks are declared shared or exclusive. The parameter s is the fraction of memory accesses found in L1 cache that references blocks that are declared shared. These shared blocks are a result of shared data and code as well as task switching.

I: The bus is used to broadcast invalidations and to determine if new blocks in L1 cache are shared or exclusive. The parameter I is the number of cycles that the bus is occupied when an invalidation is broadcast. Unlike an invalidation, a read request requires a response on the bus from the private L1 caches. This will be assumed to take an additional cycle. Therefore, a read request occupies the bus for $I+1$ cycles.

A: Only one processor can control the bus at a time. When more than one processor requests the bus simultaneously, it must be decided which will gain control. This is done with arbitration logic. When the bus is heavily used, the time needed for a bus request to pass through this logic is overlapped with the bus wait time. This parameter is needed when the bus is lightly used. In this case the request can not immediately access the bus because it must first pass through the arbitration logic. Therefore, the parameter A is the minimum number of cycles required to access the bus.

3.2. Mathematical Analysis Of The Write-Through To Main Memory Case

An approximate analytical model will be used to study the performance of the proposed two-level cache configuration. This model was first proposed in [Patel82] to analyze the performance of multiprocessors with private cache memories and shared main memory. The central idea of this model is to break up the block transfer requests into several unit requests for service. The wait time for these requests is also broken up into a series of unit requests. These unit requests are treated as random and independent. This transformation simplifies the analysis and produces quite accurate results. It was shown in that paper that the errors introduced by this approximation are less than 5% for a low miss ratio. This model is used in this section to derive equations for the write-through to main memory case. The write-back from L2 cache to main memory case will be considered later.

The chosen coherence solution causes interference in the private L1 caches due to the actions of the other processors. There are three sources of this interference. The first source is due to write requests found in a shared block in a private L1 cache. This causes an invalidation to be broadcast on the bus and occurs at the rate of $aw(1-m_1)s$ per processor cycle. It is assumed that all these invalidations are effective and that one cache is invalidated. These assumptions tend to average each other out. Not all invalidations are effective because a last remaining shared block produces ineffective invalidations, while a block can be shared by more than two caches causing more than one cache to be invalidated. This source is also assumed to produce one cycle of interference for the affected directory to be updated. Each processor receives these invalidation requests from the $N-1$ other processors, but only one out of $N-1$ requests affects a particular L1 cache. Therefore, each processor experiences $aw(1-m_1)s$ units of interference per cycle due to write requests found in shared blocks.

The second source of interference is due to write requests not found in a private L1 cache. This occurs at the rate of awm_1 and causes an invalidation to be broadcast on the bus. It is assumed that only a fraction equal to the fraction of blocks that are shared is effective and that one cache is invalidated when effective. Each processor again receives these invalidation requests from the $N-1$ other processors, but now s out of $N-1$ requests affects a particular private L1 cache. A one cycle penalty is again assumed causing each processor to experience awm_1s units of interference per cycle due to write requests not found in a private L1 cache.

The third source of interference is due to read requests not found in a private L1 cache. This causes the read request to be broadcast on the bus and occurs at the rate of $a(1-w)m_1$ per cycle. Again, only a fraction equal to s causes interference. The affected cache must signal on the bus that it holds the block and change its block's status to shared. This is assumed to take one cycle. It should be noted that the status of at most one cache block is changed since only one exclusive copy of a block can exist at a time and shared blocks do not change their status. Once again, each processor receives read requests from the $N-1$ other processors, but only s out of $N-1$ requests affects a particular private L1 cache. Therefore, each processor experiences $a(1-w)m_1s$ units of interference per cycle due to read requests not found in a private L1 cache. Let Q be the sum of the three sources of interference in the private L1 caches due to the actions of the other processors:

$$Q = aw(1-m_1)s + awm_1s + a(1-w)m_1s \tag{3.1}$$

Processor utilization U is the fraction of time that a processor spends doing useful work. This will be the main measure of performance. It will be calculated by determining its reciprocal which is the actual execution time for one useful unit of work. Let this time be Z. The time Z is composed of the useful unit of work plus the processor idle time due to cache interference and memory accesses. Each processor experiences cache interference with probability $\frac{Q}{Z}$. A processor actually experiences interference only when it is busy doing useful work, one out of Z time units. Therefore, cache interference contributes $\frac{Q}{Z^2}$ time units to time Z. To complete the equation for Z, the idle time due to memory accesses must be calculated.

The write-through, no-allocate policy was chosen for the private L1 caches so that their processors can quickly complete their write requests. It is assumed that a processor can write a word into its L1 cache buffer as well as its cache (if the word's block is present there) during the useful unit of work. The only time that a write request causes a processor to be idle, then, is when it must first gain control of the bus. This must be done for write requests to shared blocks or to blocks not found in its private L1 cache. The bus can be controlled after a request passes through the arbitration logic and waits for the bus to be unoccupied. Bus wait time will be defined to be W_b. These write requests are finished after the invalidation is broadcast on the bus. Therefore $aw(1-m_1)s + awm_1$ write requests during a cycle cause a processor to be idle for $A + W_b + I$ time units.

Each private L1 cache has only one path to the $N \times L$ crossbar. A read request not found in a private L1 cache must therefore wait if the L1 cache controller is writing a word in its buffer through to L2 cache. A read request not found in a private L1 cache is assumed to have a higher priority over pending write-throughs, but once a write-through has successfully established a path to the appropriate L2 cache module, it can not be preempted. Let this wait time that a read request incurs because of write-throughs from its own private L1 cache be W_{r1}. The read request can be issued to L2 cache after an average wait of W_{r1} time units. Due to conflicts resulting from requests of the other processors, the read request will have to wait longer before accessing the appropriate L2 cache module. This wait time will be defined to be W_2.

Each L2 cache module also has only one path to the $L \times M$ crossbar. A read request not found in a private L1 cache nor in L2 cache must wait if the L2 cache module controller is writing a word in its buffer through to main memory. Again, once a write-through has sucessfully established a path to the appropriate main memory module, it can not be preempted. Let W_{r2} be the wait time that a read request incurs due to write-throughs from L2 cache to main memory. After this wait the read request is issued to main memory. The wait time for a particular main memory module will be defined to be W_{mm}. Due to conflicts resulting from requests of the other processors, the read requests will wait on average W_{mm} more time units before accessing the appropriate main memory module.

Now, the equation for Z can be completed. It is assumed that a processor can read a word out of its private L1 cache during the useful unit of work, causing no processor idle time. If a read request is not found in its private L1 cache, the processor is idle until the word's block has been read into L1 cache. If the word's block is found in L2 cache, the processor is idle for at least $W_{r1}+W_2+T_{12}$. The read request can not be completed until the result of the bus broadcast is known. This takes $A+W_b+I+1$ time units. The processor is idle for whichever process takes the longer to complete. If the word's block is not found in L2 cache, the processor is idle an additional $W_{r2}+W_{mm}+T_{2mm}$ time units to get the word's block from main memory. During a cycle, a processor issues $a(1-w)m_1$ read requests that are not found in its private L1 cache. Of these, $1-m_2$ are found in L2 cache, while m_2 must access main memory. Therefore, the actual execution time for one useful unit of work is:

$$Z = 1 + \frac{Q}{Z^2} + (aw(1-m_1)s + awm_1)(A + W_b + I)$$

$$+ a(1-w)m_1(1-m_2)[MAX\{(W_{r1}+W_2+T_{12}),(A+W_b+I+1)\}]$$

$$+ a(1-w)m_1m_2[MAX\{(W_{r1}+W_2+T_{12}+W_{r2}+W_{mm}+T_{2mm}),(A+W_b+I+1)\}] \quad (3.2)$$

The various wait times must now be determined.

Let R_{rl2} be the probability that a read request is issued by a processor during a cycle and is not found in its private L1 cache: $R_{rl2}=a(1-w)m_1$. If a read miss occurs in a private L1 cache immediately after the first unit of service of its L1 cache controller writing through a word to L2 cache, then the read request must wait for $T_{sl12}-1$ cycles for the path to the $N \times L$ crossbar. This occurs with probability R_{rl2}. If the read miss occurs after the second unit of service, then it waits $T_{sl12}-2$ cycles. This happens with a probability $(1-R_{rl2})R_{rl2}$. Therefore, the average wait time for a read request missing in a private L1 cache due to its controller writing through a word to L2 cache is:

$$\sum_{1 \leqslant i \leqslant T_{sl12}} R_{rl2}(1-R_{rl2})^{i-1}(T_{sl12}-i).$$

During a processor cycle, aw write requests contribute the above average wait time to R_{rl2} read requests. Therefore, the average wait time incurred by a read request missing in a private L1 cache due to write-throughs from its controller to L2 cache is:

$$W_{r1}=aw \sum_{1 \leqslant i \leqslant T_{sl12}} (1-R_{rl2})^{i-1}(T_{sl12}-i) \tag{3.3}$$

W_{r2} can be similarly calculated. Let R_{rmm} be the probability that an L2 cache module receives a read request during a cycle and that it is not found there. N processors each produce read requests during a cycle with probability $a(1-w)$. These are not found in a private L1 cache with probability m_1 and are equally distributed among the L L2 cache modules. Each L2 cache module therefore receives read requests during a cycle with probability $\frac{Na(1-w)m_1}{L}$, so $R_{rmm}=\frac{Na(1-w)m_1m_2}{L}$. The average wait time for a read request missing in L2 cache due to write-throughs to main memory is:

$$\sum_{1 \leqslant i \leqslant T_{slmm}} R_{rmm}(1-R_{rmm})^{i-1}(T_{slmm}-i)$$

Each of the N processors issue aw write requests during a cycle which are equally distributed among the L L2 cache modules, so each L2 cache module executes $\frac{Naw}{L}$ write-throughs per cycle. These contribute the above wait time to R_{rmm} read requests. Therefore, the average wait incurred by a read request not found in a private L1 cache nor in L2 cache due to write-throughs to main memory is:

$$W_{r2}=\frac{Naw}{L}\sum_{1\leqslant i\leqslant T_{s1mm}}(1-R_{rmm})^{i-1}(T_{s1mm}-i) \tag{3.4}$$

The remaining unknown wait times will be determined by finding the bandwidths of main memory, L2 cache, and the bus using two different methods. Main memory is used to transfer blocks to L2 cache and to complete write-throughs. Blocks are transferred to L2 cache whenever one of the N processors issues a read request not found in its private L1 cache nor in L2 cache. All write requests from the N processors eventually are written into main memory. Therefore, the bandwidth of main memory is:

$$B_{mm}=\frac{Na(1-w)m_1m_2T_{2mm}}{Z}+\frac{NawT_{s1mm}}{Z} \tag{3.5}$$

Another way to determine the bandwidth of main memory is to calculate the average number of busy main memory modules. This is done by first determining the main memory request rate from each L2 cache module. As can be seen above, an L2 cache module requests service from main memory for block transfers and write-throughs. The wait time for a particular main memory module W_{mm} is also part of the request rate. The total number of requests to main memory comes equally from each of the L L2 cache modules. Therefore, the unit request rate for main memory from each L2 cache module is:

$$u_{12}=\frac{Na(1-w)m_1m_2(W_{mm}+T_{2mm})}{LZ}+\frac{Naw(W_{mm}+T_{s1mm})}{LZ} \tag{3.6}$$

All the requests to main memory are equally divided among the M main memory modules. Each main memory module then receives a request from an L2 cache module with probability $\frac{u_{12}}{M}$. The probability that none of the L L2 cache modules are requesting a particular main memory module is: $(1-\frac{u_{12}}{M})^L$. Therefore, the probability that an L2 cache module is requesting a particular main memory module is: $1-(1-\frac{u_{12}}{M})^L$. The average number of busy main memory modules is then the number of main memory modules times the above probability:

$$B_{mm}=M[1-(1-\frac{u_{12}}{M})^L] \tag{3.7}$$

Substituting (3.6) into (3.7) and equating it to (3.5) produces an equation with which W_{mm} can be determined when the value of Z is known.

The wait time for a particular L2 cache module is determined similarly. L2 cache is used to transfer blocks to L1 cache and to store write-throughs from the L1 cache controllers. If a read request is found in L2 cache, the cache module is occupied for the duration of the block transfer. If not, the cache module is occupied for the block transfer as well as the wait times for the path to main memory and for a particular main memory module and the transfer time from main memory to L2 cache for a block. All write requests pass through L2 cache. Therefore, the bandwidth of L2 cache is:

$$B_{12}=\frac{Na(1-w)m_1(1-m_2)T_{12}}{Z}+\frac{Na(1-w)m_1m_2(W_{r2}+W_{mm}+T_{2mm}+T_{12})}{Z}+\frac{NawT_{s112}}{Z} \tag{3.8}$$

The wait time for a particular L2 cache module is also included in the unit request rate for L2 cache from each L1 cache controller:

$$u_{11}=\frac{a(1-w)m_1(1-m_2)(W_2+T_{12})}{Z}+\frac{a(1-w)m_1m_2(W_2+W_{r2}+W_{mm}+T_{2mm}+T_{12})}{Z}$$
$$+\frac{aw(W_2+T_{s112})}{Z} \tag{3.9}$$

The requests to L2 cache are equally distributed among the L L2 cache modules causing each to receive a request from an L1 cache controller with probability $\frac{u_{11}}{L}$. The probability that none of the N L1 cache controllers are requesting a particular L2 cache module is $(1-\frac{u_{11}}{L})^N$. The probability that a particular L2 cache module is being used is $1-(1-\frac{u_{11}}{L})^N$. The bandwidth of L2 cache is therefore:

$$B_{12}=L[1-(1-\frac{u_{11}}{L})^N] \tag{3.10}$$

W_2 can be determined when the values of Z and W_{mm} are known by substituting (3.9) into (3.10) and equating it to (3.8).

The bus is used to broadcast invalidations when write requests reference shared blocks or when write requests are not found in a private L1 cache. Read requests are broadcast when they are not found in a private L1 cache. An invalidation occupies the bus for I cycles, while a read request requires an extra cycle. All N processors use the same bus. Therefore, the bus bandwidth is:

$$B_{bus}=\frac{Naw(1-m_1)sI}{Z}+\frac{Nawm_1I}{Z}+\frac{Na(1-w)m_1(I+1)}{Z} \tag{3.11}$$

The bus unit request rate from each processor also includes the wait time W_b that a request experiences when the bus is busy:

$$u_b=\frac{aw(1-m_1)s(W_b+I)}{Z}+\frac{awm_1(W_b+I)}{Z}+\frac{a(1-w)m_1(W_b+I+1)}{Z} \tag{3.12}$$

There is only one bus, so the probability that none of the N processors are requesting the bus is $(1-u_b)^N$. The probability that the bus is being used is the bus bandwidth:

$$B_{bus}=1-(1-u_b)^N \tag{3.13}$$

The bus wait time can be determined if Z is known by substituting (3.12) into (3.13) and equating it to (3.11). Since Z must be known to calculate the wait times and the wait times must be known to calculate Z, these variables are found by using numerical analysis.

3.3. Discussion of Results

The results of the analysis of the previous section are presented in this section. Because of the large number of parameters, a default value will be chosen for each. Then the effect of each parameter can be studied individually by changing its value while keeping the others at their default value. The values of most parameters can vary over a large range. The chosen default values are as follows:

$L=16$ If L2 cache is the bottleneck of the system, it can be divided into more modules. This increases the possible bandwidth of L2 cache. Since this can be done, a large number of L2 cache modules will be assumed. For individual cases, it is fairly easy to find the smallest number of L2 cache modules needed. This will be done for the default parameters. The reason that the smallest possible number of modules should be used is that the smaller the number, the smaller the crossbar switches. Large crossbar switches are slow and expensive.

$M=16$ The arguments for the number of L2 cache modules also apply to main memory. Again, it is fairly easy to determine the smallest number of main memory modules needed for a specific case and this will be done for the default parameters.

$m_1=10\%$ The private L1 caches have been assumed to be small and fast. Therefore, their miss ratio is fairly high. The chosen value of 10% should be the largest allowable miss ratio. Higher miss ratios will not provide very good performance. Lower miss ratios should be possible, so the values 7.5%, 5%, and 2.5% will also be looked at.

$M_2=1\%$ Shared L2 cache should be quite large as it is shared by all the processors. Large caches can have very low miss ratios. The value of 1% has been chosen as an average value. Shared L2 cache miss ratios of 2%, 5%, and 0.1% will also be studied.

$T_{12}=5$ This parameter can have a large variation depending on the block size and bandwidth of the $N \times L$ crossbar. L2 cache is slower than L1 cache so more than one cycle is required for the transfer. The value of 5 is again assumed to be an average value and values of 2, 8, and 10 will also be analyzed.

$T_{2mm}=20$ Main memory is significantly slower than the caches. This is why the caches are used in the first place. The blocks transferred between main memory and L2 cache will usually be bigger than those transferred to L1 cache. Therefore, this transfer time is much longer than T_{12}. Again, T_{2mm} can have a wide range of values, so values of 10, 30, and 40 will also be studied.

$T_{sl12}=2$ A block will usually contain many words, so a single word transfer is faster than a block transfer. A word should be small enough that it can all be transferred at once. Therefore, the word transfer time to L2 cache is taken to be only one more cycle than to L1 cache due to the fact that L2 cache is slower. Larger values will also be studied to see their effect.

$T_{slmm}=3$ Main memory is slower than L2 cache, so a one word write into main memory is assumed to require one more cycle than T_{sl12}. Again, larger values will be substituted to see their effect.

$a=90\%$ The memory access rate is very dependent on program behavior. Since some tasks tend to be processor-bound while others are memory-bound, this parameter can vary over a large range. The pessimistic value of 90% has been chosen to show that the proposed configuration gives good performance even when mostly memory-bound tasks are being run on it. The effect of this parameter will be shown by also looking at the values of 80% and 100% as well as 50%.

$w = 20\%$ [Smith82] shows the percentages of memory accesses that result from write requests for a number of program traces. This value ranges from 5% to 34% with an overall average of 16%. Most results are close to the average, so this parameter will not be varied.

$s = 5\%$ The degree of sharing can vary over a large range. It can be quite high if a single task is running on multiple processors or a large amount of task switching is taking place. If a different task is occupying each processor, then very little communication is needed among them and s is very small. This second case will be taken as the norm and the value 5% chosen as the default value. Since s can vary significantly, the values 15% and 50% (more representative of the first case) as well as 1% will be studied.

$I = 2$ The number of cycles that the bus is used for an invalidation broadcast depends on how the requests are handled. If the requests are required to remain on the bus until all the processors have taken care of them, the bus can be occupied for many cycles per request. In the case of a broadcasted read request, the processors are required to respond over the bus if they hold the requested word. This response can be overlapped with other requests. If this is done, the bus can be more efficiently utilized. For the default case, it is assumed that the processors quickly respond to broadcasted requests and that the bus is occupied until the request is completed, two cycles for invalidations and three cycles for read requests. To study the effects of other implementations, the values of 1, 3, and 4 will be analyzed.

$A = 1$ It is important that the bus arbitration logic causes as little interference to the system as possible. This logic should determine the type and source of each request it receives during each cycle and signal who is allowed to use the bus during the next cycle if it will be unoccupied. This parameter is therefore taken to be one cycle and will not be varied.

In all cases, the number of processors will be varied from 2 to 20. A number of the parameters will vary with the number of processors in the system. The shared L2 cache miss ratio will increase with an increasing number of processors because each processor must use smaller portions of L2 cache. More processors increase the chances of a block being shared. Also, the bus must be longer for more processors making broadcast time longer. These effects will noι be considered in this analysis. Therefore, the performance measured for large numbers of processors will probably be slightly optimistic.

The effects of varying the private L1 cache miss ratio are shown in Figures 3-1 through 3-3. Figure 3-1 shows the system performance of the four private L1 cache miss ratios studied. System performance is the total processor utilization in the system found by multiplying the number of processors in the system and the utilization of each processor. In each case, the system performance increases linearly with an increasing number of processors and then levels off at its maximum value. It can be seen in Figure 3-2 that the number of processors where the system performance levels off corresponds to the minimum number of processors that saturate the bus. It takes about eight processors to saturate the bus for the default case of $m_1 = 10\%$. The system performance increases almost linearly with an increasing number of processors until $N = 8$ and then levels off. The bus is the bottleneck of the system. Adding more processors beyond this point does not improve the system performance because the utilization of each processor drops almost linearly with an

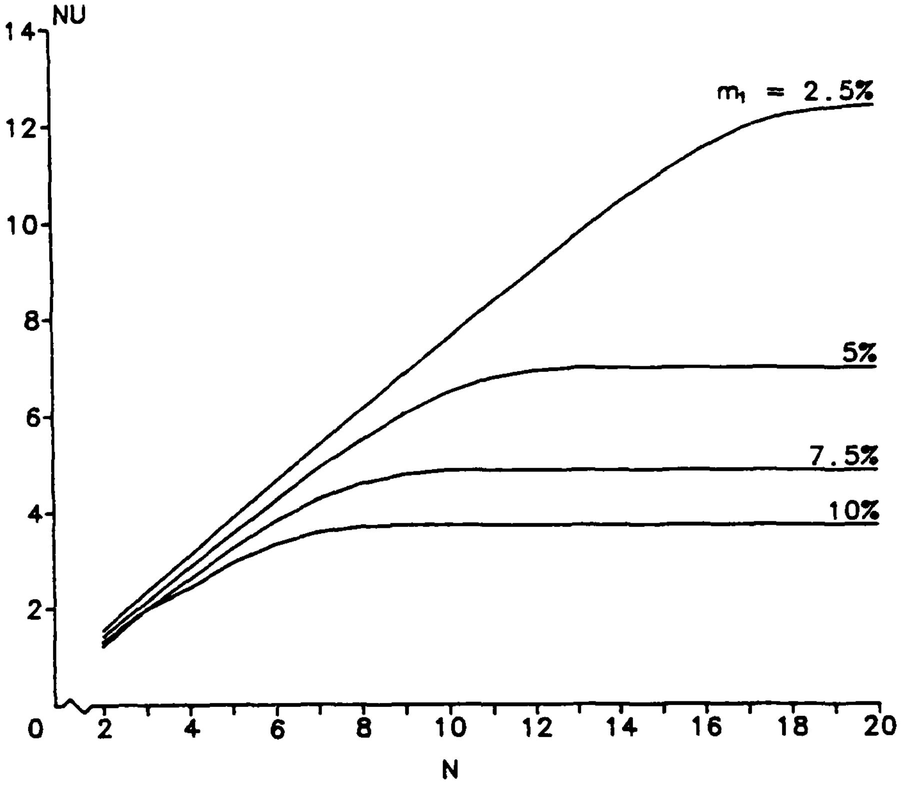


Figure 3-1. System performance vs. number of processors. Effect of private L1 cache miss ratio on system performance.

increasing number of processors. This can be seen in Figure 3-3. The processor utilization decreases only slightly for an increasing number of processors due to increasing memory conflicts until the bus is saturated. Then it drops almost linearly. Therefore, the optimum number of processors in the system is the smallest number that saturates or nearly saturates the bus. It can be seen from the three figures that the smaller the private L1 cache miss ratio, the better the system performance. Only eight processors can be efficiently utilized for m_1 = 10%, while this number is 10 for m_1 = 7.5%, 13 for m_1 = 5%, and 19 for m_1 = 2.5%.

The bus is the bottleneck for all cases looked at so far. This is because the default values for L and M were chosen so that this would happen. The results for those cases

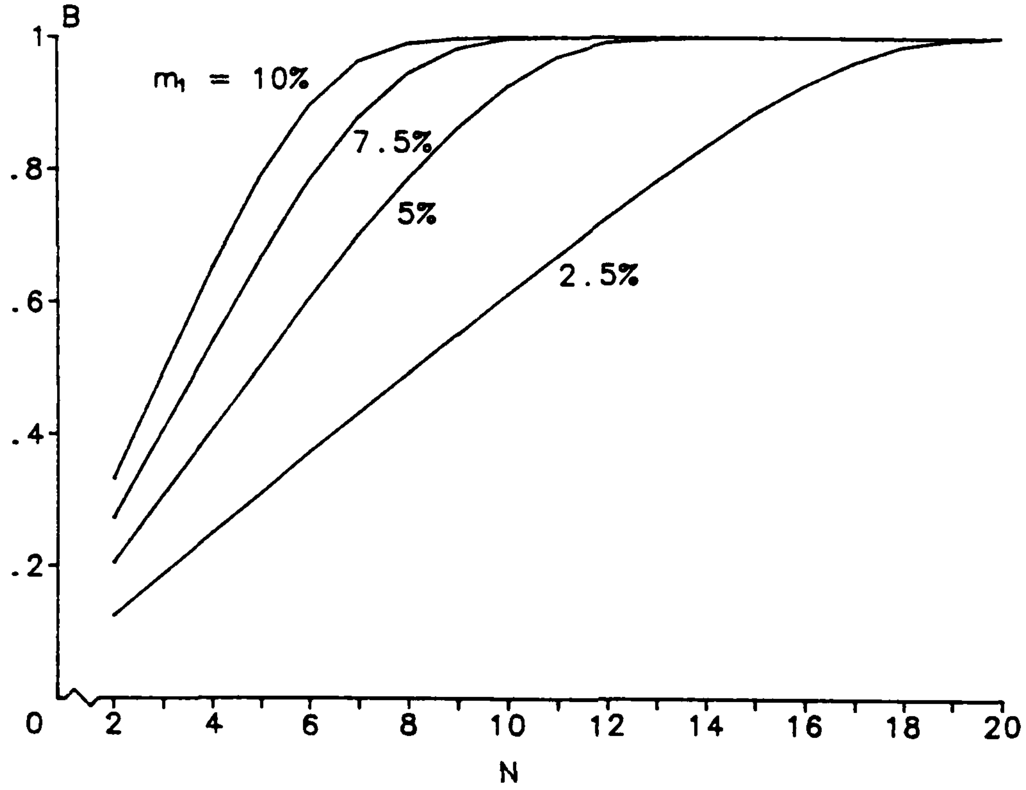


Figure 3-2. Bus utilization vs. number of processors. Effect of private L1 cache miss ratio on bus utilization.

show that the maximum L2 cache bandwidth needed is only 3.25 modules and that the maximum main memory bandwidth needed is 2.55 for the default case. This implies that only four L2 cache modules and three main memory modules are needed. The number of modules was changed to $L = 4$ and $M = 3$ and main memory was found to be a severe bottleneck. This somewhat surprising result is attributable to Equation (3.7) of the previous section: $B_{mm} = M[1-(1-\frac{u_{12}}{M})^L]$. When main memory is the bottleneck of the system, u_{12} approaches 1. Therefore, the bandwidth of main memory with $L = 4$, $M = 3$, and $u_{12} = 1$ is 2.41 which is not enough. The bandwidth of main memory with $L = 4$, $M = 4$, and $u_{12} = 1$ is 2.73 which is enough. The number of modules used was changed to $L = 4$ and

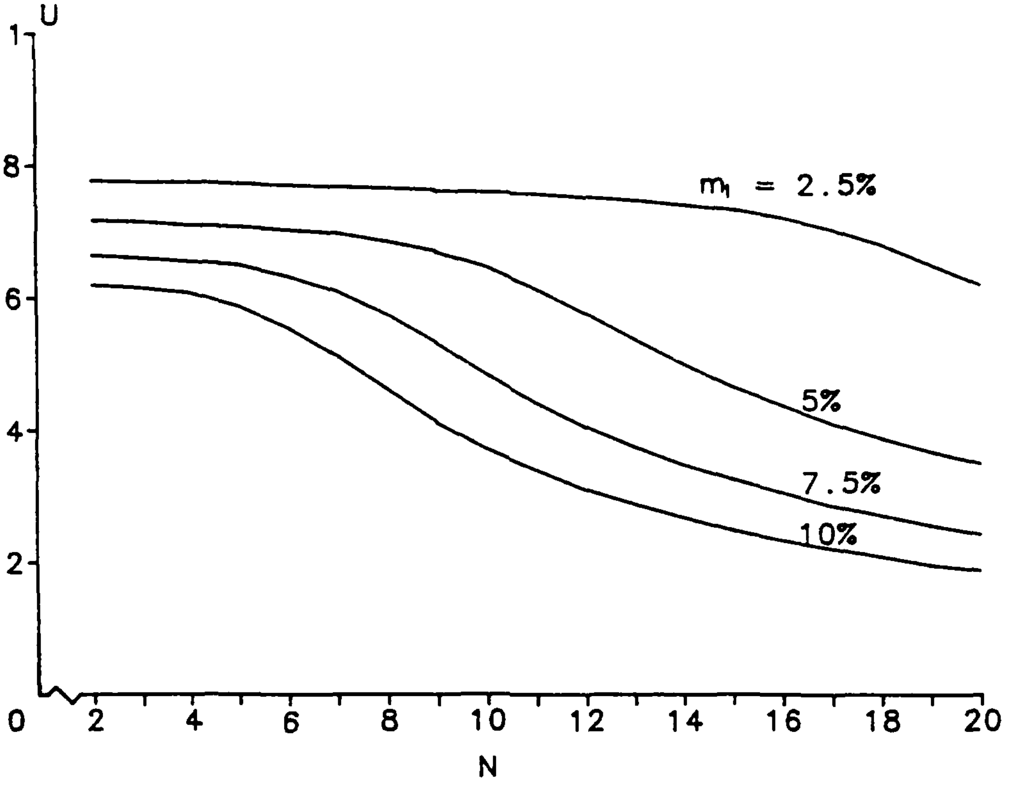


Figure 3-3. Processor utilization vs. number of processors. Effect of private L1 cache miss ratio on processor utilization.

M = 4 and the bus was found to be the bottleneck, as predicted. The system performance using L = 4 and M = 4 was in the worst case only 3.5% below the system performance using L = 16 and M = 16. Therefore, the most cost-effective configuration for the default case is N = 8, L = 4, and M = 4. The number of modules was also changed to L = 3 and M = 4 to see if L2 cache becomes the bottleneck. It did as predicted. This case in fact gave the worst system performance of all. The results for the various values of L and M are shown in Figure 3-4. This figure also shows the system performance when all wait times are zero and no coherence solution is used. The chosen coherence solution does not degrade system performance too much until the bus becomes saturated. Up to that point, the main difference in system performance between the ideal case and the chosen coherence solution

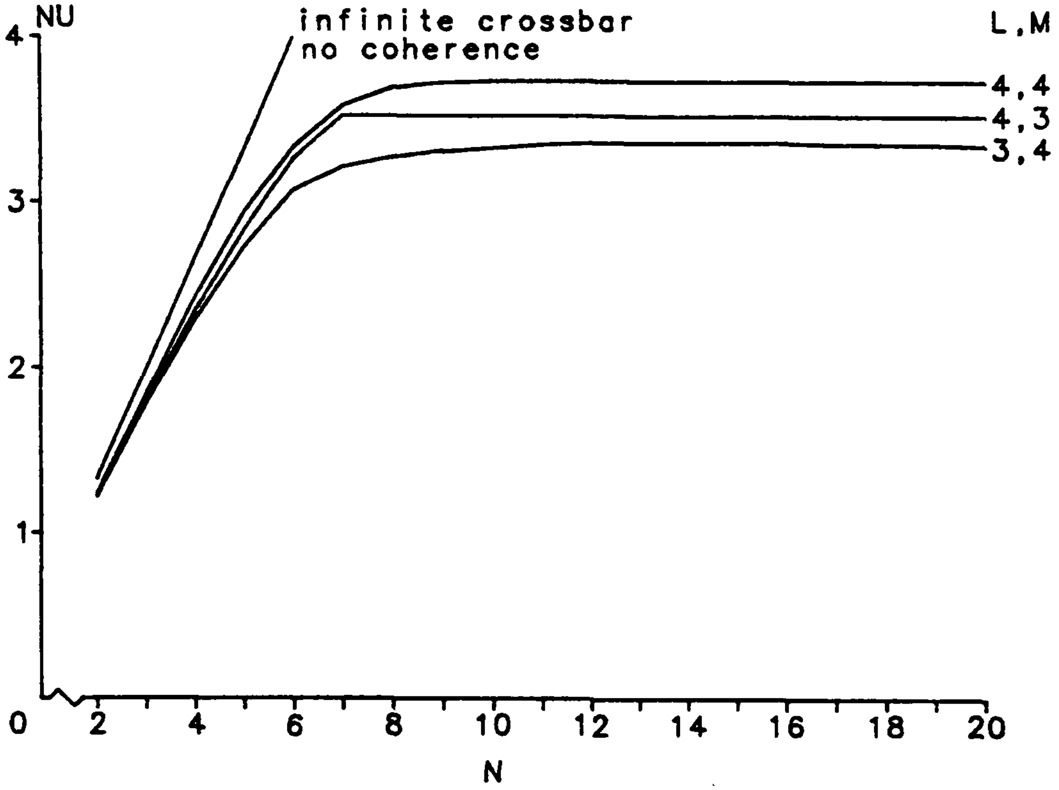


Figure 3-4. System performance vs. number of processors. Effect of number of L2 cache and main memory modules on system performance.

is the non-zero wait times calculated by the model.

Figure 3-5 shows the system performance for the various values of the shared L2 cache miss ratio. As in all the remaining cases to be looked at, the bus is the bottleneck. M_2 only indirectly affects the bus utilization through the value of Z. A decrease in M_2 causes Z to be smaller and the bus to saturate for a smaller number of processors. The opposite, of course, occurs for an increase in M_2. The value of M_2 is very significant to the system performance when the number of processors is such that the bus is not yet saturated. As expected in this case, the smaller the miss ratio the better the system performance. An added benefit of a smaller L2 cache miss ratio is that fewer L2 cache modules and main memory modules are needed. For the worst case value of M_2=5%, a bandwidth

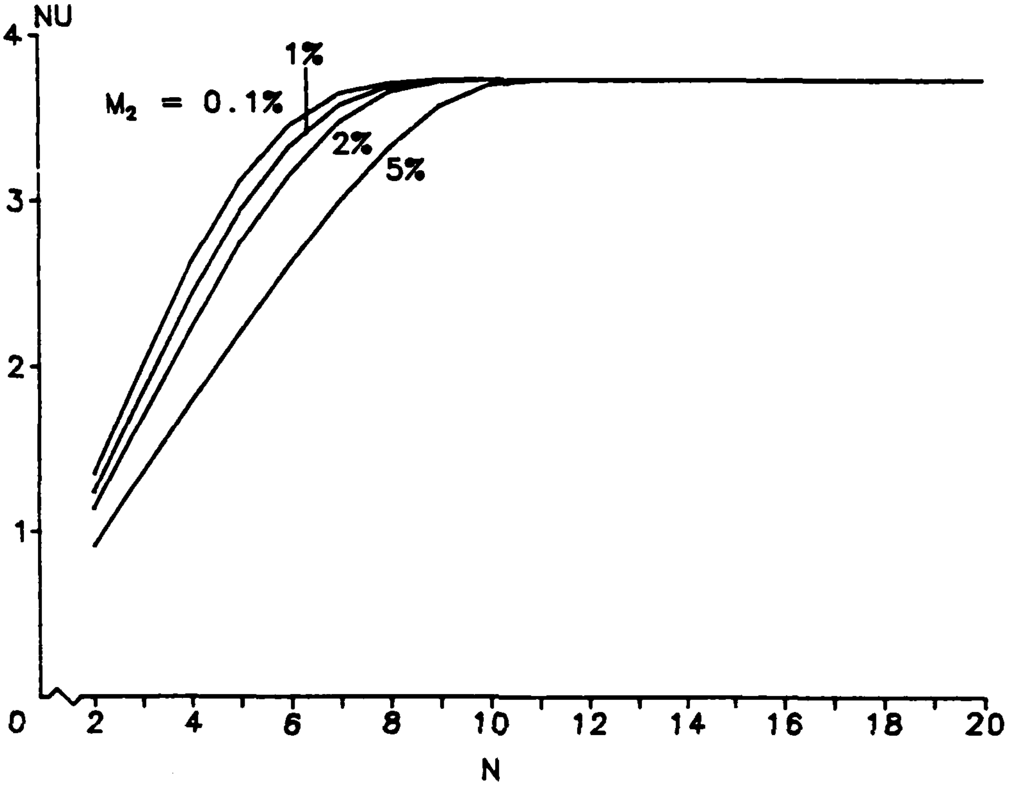


Figure 3-5. System performance vs. number of processors. Effect of shared L2 cache miss ratio on system performance.

of 5.55 for L2 cache and 4.70 for main memory is needed. For the best case value of M_2=0.1%, only a bandwidth of 2.74 for L2 cache and 2.07 for main memory is needed.

The block transfer times also indirectly affect the number of processors that cause the bus to saturate. Shorter transfer times make Z smaller and cause the bus to saturate for a smaller number of processors. Also, since Z is smaller, the system performance is larger. Again, this is only significant when the bus is not saturated. Figure 3-6 shows the effect on system performance of varying T_{12}, the number of cycles to transfer a block from L2 cache to L1 cache. The effect on system performance of varying T_{2mm}, the block transfer time from main memory to L2 cache, is shown in Figure 3-7. In addition to better system performance, shorter transfer times require fewer L2 cache and main memory modules.

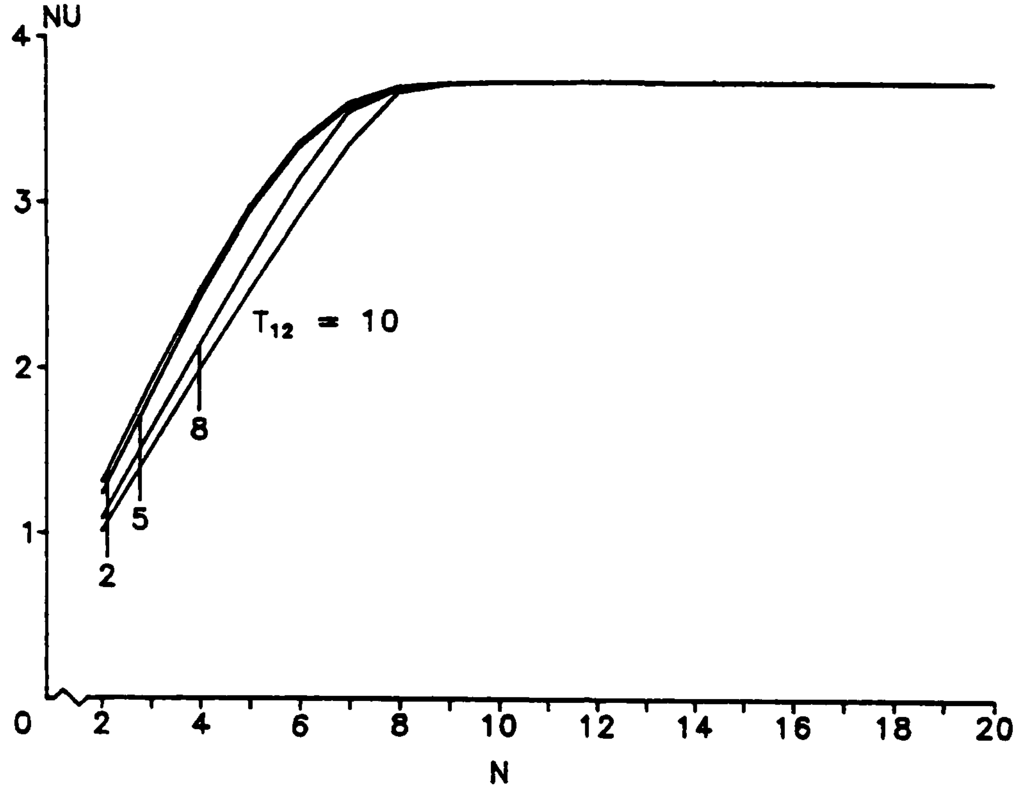


Figure 3-6. System performance vs. number of processors. Effect of block transfer time from L2 cache to L1 cache on system performance.

T_{12} affects only the bandwidth of L2 cache. The needed bandwidth of L2 cache is 4.57 for the worst case value of T_{12}=10, while it is 2.43 for the best case value of T_{12}=2. T_{2mm} affects the bandwidths of both L2 cache and main memory. For the worst case value of T_{2mm}=40, the needed bandwidth of L2 cache is 3.78 and is 3.09 for main memory. For the best case value of T_{2mm}=10, 2.97 is the needed bandwidth of L2 cache and 2.28 is the needed bandwidth of main memory.

Varying both the write times for one word into L2 cache and into main memory affects the overall system performance only slightly. The largest percentage difference between the worst case value of T_{s1l2}=5 and the best case value of T_{s1l2}=1 is 7.8%. The effect of T_{s1mm} is even less. The largest percentage difference between the worst case value

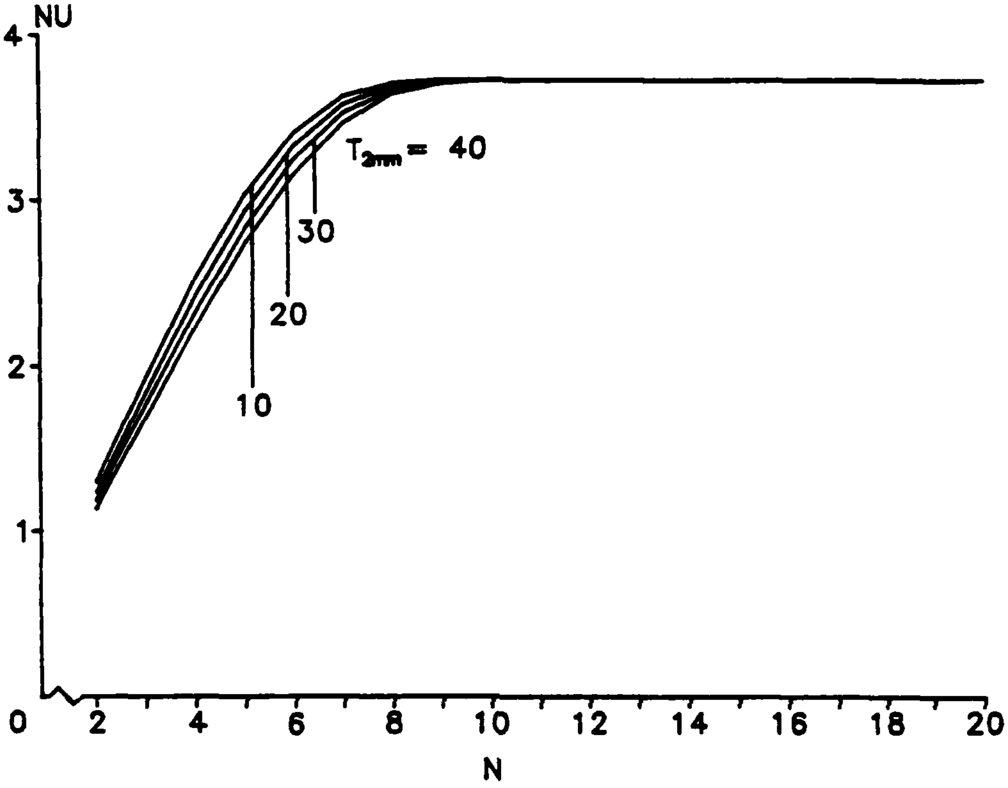


Figure 3-7. System performance vs. number of processors. Effect of block transfer time from main memory to L2 cache on system performance.

of T_{s1mm}=10 and the best case value of T_{s1mm}=2 is only 1.5%. It should be noted that the buffer sizes have been assumed to be infinite. For a finite buffer size, longer store times increase the probability that a buffer is full and that a write request must wait until space is available. Because of this, short write times are desirable even though they seem to have little effect on the results obtained.

The memory access rate, a, directly affects almost every equation developed in the previous section, including the bus bandwidth. Therefore, varying the parameter a should produce a large variation in the system performance. This can be seen to be true in Figure 3-8. The 10% difference between a =90% and a =80% causes the maximum system performance to increase by 12.5%. Unfortunately, this parameter is not affected by changes in

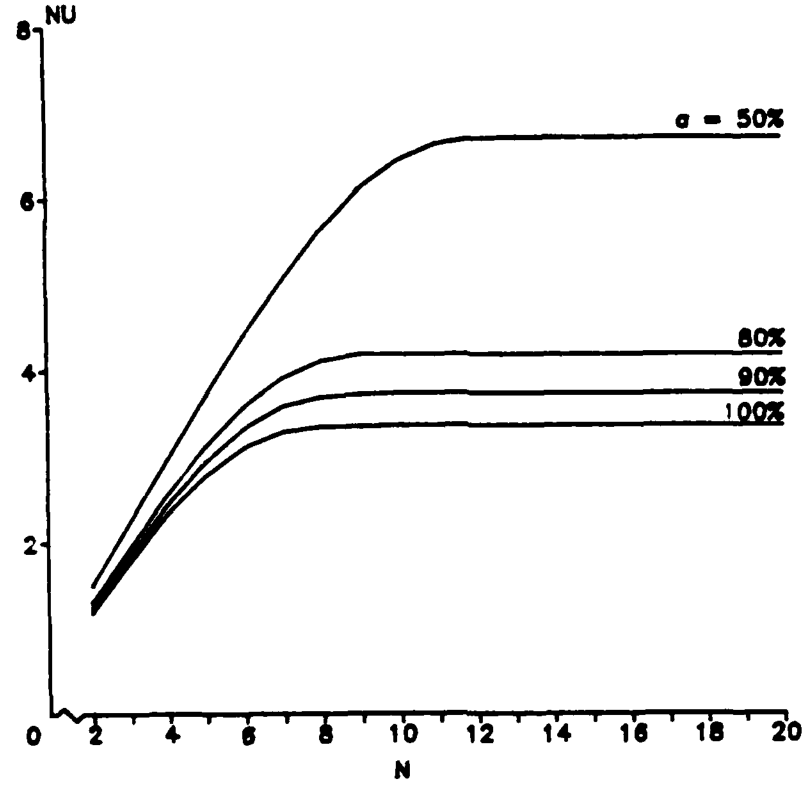


Figure 3-8. System performance vs. number of processors. Effect of memory access rate on system performance.

the memory configuration. This parameter shows the need for processors which have instruction sets that make good use of internal registers for use in multiprocessor systems.

Shared blocks are the source of the coherency problems in the private L1 caches. It is not surprising, then, that the degree of sharing has a direct effect on the system performance. The parameter *s* is the fraction of write requests that are found in L1 cache that must use the bus. More shared blocks mean more bus usage and less system performance. This effect is shown in Figure 3-9. The degree of sharing can be controlled somewhat by limiting the effect of task switching. This implies that the operating system should be written to try to return tasks to the same processor that they had previously occupied as much as possible.

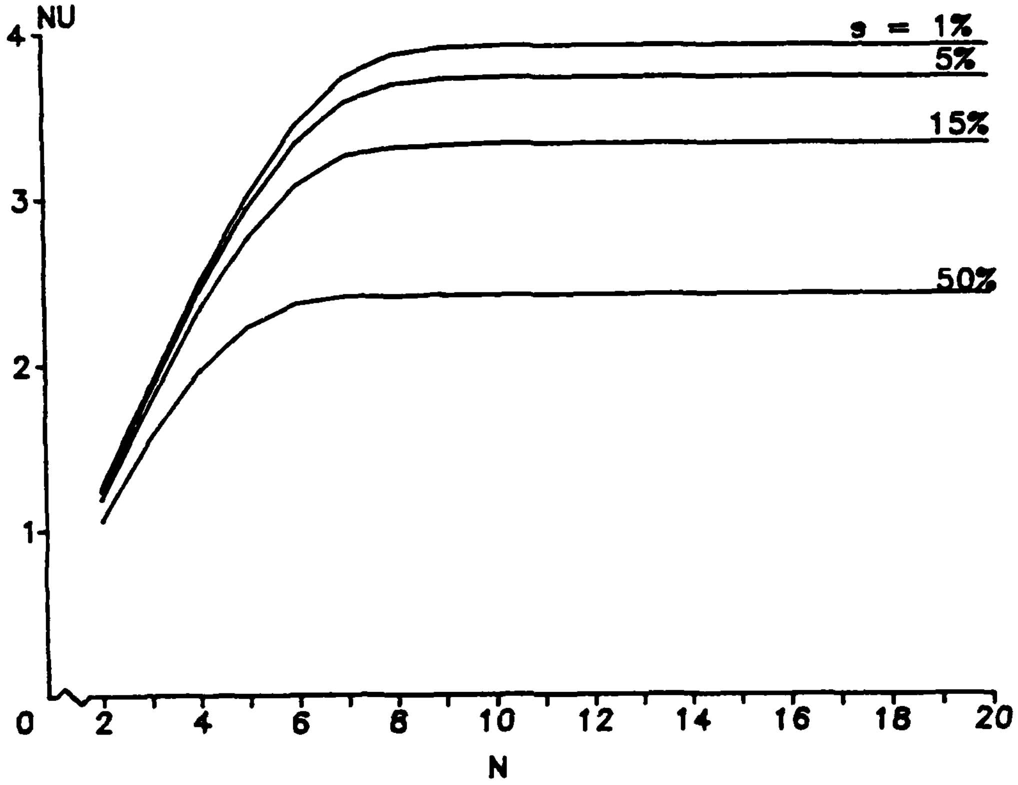


Figure 3-9. System performance vs. number of processors. Effect of degree of sharing on system performance.

The final parameter to be studied is the one that affects the bus utilization most directly: the number of cycles that the bus is occupied for an invalidation broadcast, I. Figure 3-10 shows the effect of this parameter. It can be seen that doubling the broadcast time from $I=1$ to $I=2$ nearly cuts the maximum system performance in half. It is therefore crucial that the broadcast time be as short as possible.

The equations in the previous section were modified to model the write-back to main memory from L2 cache case. A new parameter was found to be needed to define the percentage of replaced L2 cache blocks that must be written back to main memory. Numerical analysis was performed on this new set of equations and it was found that the system performance was nearly identical to the write-through to main memory case. This was true

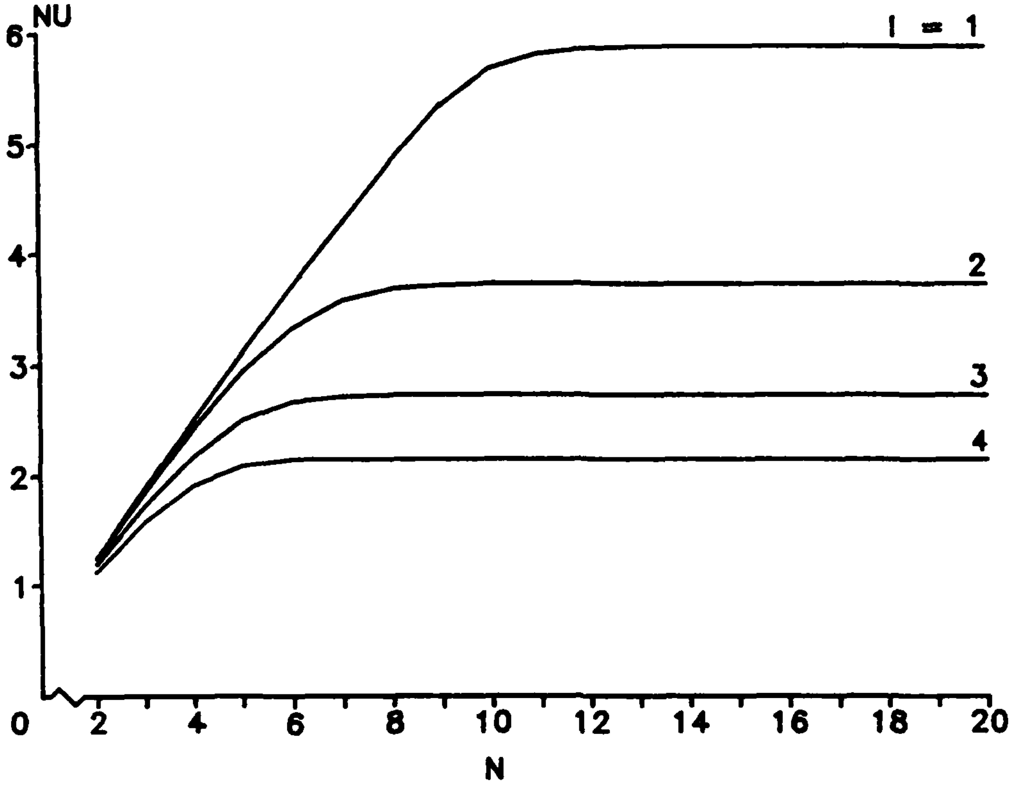


Figure 3-10. System performance vs. number of processors. Effect of bus broadcast time on system performance.

over the full range of the new parameter used. Since neither L2 cache nor main memory was the bottleneck in either case, this result was not very surprising. The two write policies to main memory were then compared using the minimum module numbers of L =4 and M =4. Again, the system performance results did not vary significantly. Using the write-back policy to main memory did reduce the needed main memory bandwidth, though. When half the replaced L2 cache blocks must be written back to main memory, the maximum required bandwidth for main memory was only 1.01. Fewer main memory modules are therefore needed for the write-back to main memory case. The choice of write policies to main memory is still a tough choice, though, as the write-through policy provides better system reliability. Main memory always contains valid copies of the blocks it holds. If a cache fails, its data is not lost because it has been updated in main memory. Also, any error detected in a cache can be corrected by finding the corresponding word in main memory.

CHAPTER 4

A METHOD TO REDUCE BUS USAGE

The previous chapter analyzed a coherence solution for the proposed two-level cache configuration and the bus was found to be the bottleneck of the system. This chapter proposes a change to this coherence solution designed to reduce the usage of the bus and increase th- system performance. Using the default parameters, 80.5% of the bus usage is for broadcasting read requests. A reduction in the number of read requests that are required to be broadcast on the bus will therefore have a large impact on the bus usage. This can be accomplished by adding directory information to L2 cache. If L2 cache knows what blocks are presently in L1 cache, then read requests not found in L1 cache can check L2 cache to see if the block will be shared or exclusive. A read request missing in a private L1 cache is then broadcast on the bus only when it is found in L2 cache that another private L1 cache already holds a copy of it. This broadcast is required only to change the status of the other copy of the block since it has probably been declared an exclusive block. No response is then required for read requests, so they no longer take one more cycle than invalidation broadcasts.

It is important that the advantages gained by using directory information in L2 cache are not outweighed by disadvantages somewhere else. The directory information must be accessible and changeable without much degradation in system performance. The easiest way to keep track of what is in L1 cache is to require it to be a subset of what is in L2 cache. Each block in L2 cache only needs a single bit to indicate whether or not it is present in L1 cache. It is not important to know which private L1 cache(s) holds a particular block. The requirement that L1 cache be a subset of L2 cache is not difficult to satisfy. All L1 cache blocks come from L2 cache. The only problem is when L2 cache blocks are replaced. When this occurs, any remaining child L1 blocks of the replaced L2 block are invalidated. This seems to be counterproductive since invalidating additional L1 blocks increases the bus usage, increases the L1 cache miss ratio, and increases cache interference. It will be shown that these disadvantages are less than the advantages gained for a large number of processors. It may be impractical to require L1 cache to be a subset of L2 cache. A large number of processors requires L2 cache to be very large. This makes it expensive and slow. To alleviate this problem, directory information can be held for a number of the least recently used blocks without their data. Requests to these blocks will be treated as misses except for determining the presence of a particular block in L1 cache.

To determine the performance of this subset requirement, a new parameter must be introduced. This parameter is the percentage of read requests found in L2 cache that are already present in a different private L1 cache than the one making the request. This parameter will be called p and can also be defined as the percentage of L2 cache blocks that are read out to L1 cache more than once. A block is read out of L2 cache more than once either because it is used by more than one processor, due to task switching or being a shared block, or because the block is replaced in L1 cache and then requested again. It is difficult to predict the value of this parameter, so it will be studied over its full range.

Most of the equations derived in Chapter 3 apply here, too, either as is or with slight changes. The additional source of cache interference must be added to the equation for Q. L2 cache blocks are replaced when a read request is not found in a private L1 cache nor in L2 cache at a rate of $a(1-w)m_1m_2$ per processor cycle. These replacements now cause invalidations to be broadcast on the bus. It is assumed that all these invalidations are effective and that one cache is invalidated. The invalidations cause one cycle of interference for the affected cache. N processors produce the read requests that eventually produce these invalidations, but only one out of the N L1 caches is affected by an invalidation.

Therefore, these invalidations contribute an extra $a(1-w)m_1m_2$ cycles of cache interference to each processor:

$$Q=aw(1-m_1)s+awm_1s+a(1-w)m_1s+a(1-w)m_1m_2 \tag{4.1}$$

The subset requirement was proposed to reduce bus usage, so the bus utilization equations must all be modified. Equation (3.11) must be both modified and added to. Read requests now occupy the bus for just I cycles, and only p of the read requests found in L2 cache must be broadcast on the bus. Read requests not found in L2 cache do not need to be broadcast since it is now guaranteed that the block is not in L1 cache and the new block will be declared exclusive. Broadcast invalidations for replaced L2 blocks contribute $\dfrac{Na(1-w)m_1m_2I}{Z}$ to the bus usage. These changes are shown in the following equation:

$$B_{bus}=\frac{Naw(1-m_1)sI}{Z}+\frac{Nawm_1I}{Z}+\frac{Na(1-w)m_1(1-m_2)pI}{Z}+\frac{Na(1-w)m_1m_2I}{Z} \tag{4.2}$$

The unit request rate for the bus is now split between the processors and the L2 cache modules. The invalidations broadcast for replaced L2 cache blocks come from the L L2 cache modules equally, so the unit request rate per L2 cache module for the bus is:

$$u_{bc}=\frac{Na(1-w)m_1m_2(W_b+I)}{LZ} \tag{4.3}$$

The remainder of the bus requests are made by the N processors:

$$u_{b_p}=\frac{aw(1-m_1)s(W_b+I)}{Z}+\frac{awm_1(W_b+I)}{Z}+\frac{a(1-w)m_1(1-m_2)p(W_b+I)}{Z} \tag{4.4}$$

Equation (3.13) must be slightly modified. The probability that none of the N processors are requesting the bus is $(1-u_{b_p})^N$. The probability that none of the L L2 cache modules are requesting the bus is $(1-u_{bc})^L$. The probability that the bus is not being requested is the product of the two probabilities above. Therefore, the bus utilization is:

$$B_{bus}=1-(1-u_{b_p})^N(1-u_{bc})^L \tag{4.5}$$

The real execution time for one useful unit of work must also be modified. Read requests not found in a private L1 cache nor in L2 cache do not use the bus, so the second MAX function of Equation (3.2) can be removed along with its second option. Only read requests found in L2 cache with their presence bit set are required to use the bus. The remaining $1-p$ read requests simply access L2 cache, find that the block is not present in L1 cache, update the status bit, and read the block into the requesting private L1 cache. This occurs frequently since many L1 cache blocks are brought into L2 cache for a block transfer from main memory. Write requests are handled the same way as in Chapter 3. The changes are shown in the following equation:

$$
\begin{aligned}
Z = 1 + \frac{Q}{Z^2} &+ (aw(1-m_1)s + awm_1)(A + W_b + I) \\
&+ a(1-w)m_1(1-m_2)(W_{r1} + W_2 + (1-p)T_{12} + p[MAX\{(T_{12}),(A + W_b + I)\}] \\
&+ a(1-w)m_1 m_2(W_{r1} + W_2 + T_{12} + W_{r2} + W_{mm} + T_{2\mathrm{mm}})
\end{aligned}
\tag{4.6}
$$

It is seen that read requests found in L2 cache with its block's presence bit set take longer to complete than a similar request in the previous solution. This is because L2 cache must be accessed before the request is sent to the bus. The remaining equations in Chapter 3 apply here unmodified.

Figure 4-1 shows the system performance obtained by keeping L1 cache a subset of L2 cache. The effect of the higher L1 cache miss ratio has been modeled by making m_1 = 15%. Slightly worse system performance results when the number of processors in the system is six or less. The bus is not yet saturated, so the benefits of the subset requirement are not great enough to offset the increased L1 cache miss ratio and the increased time needed to complete read requests found to be shared in L2 cache. For more than six processors, the subset requirement provides better system performance if the percentage of L2 cache blocks whose presence bit is set is less than 90%. It seems reasonable to assume that realistic

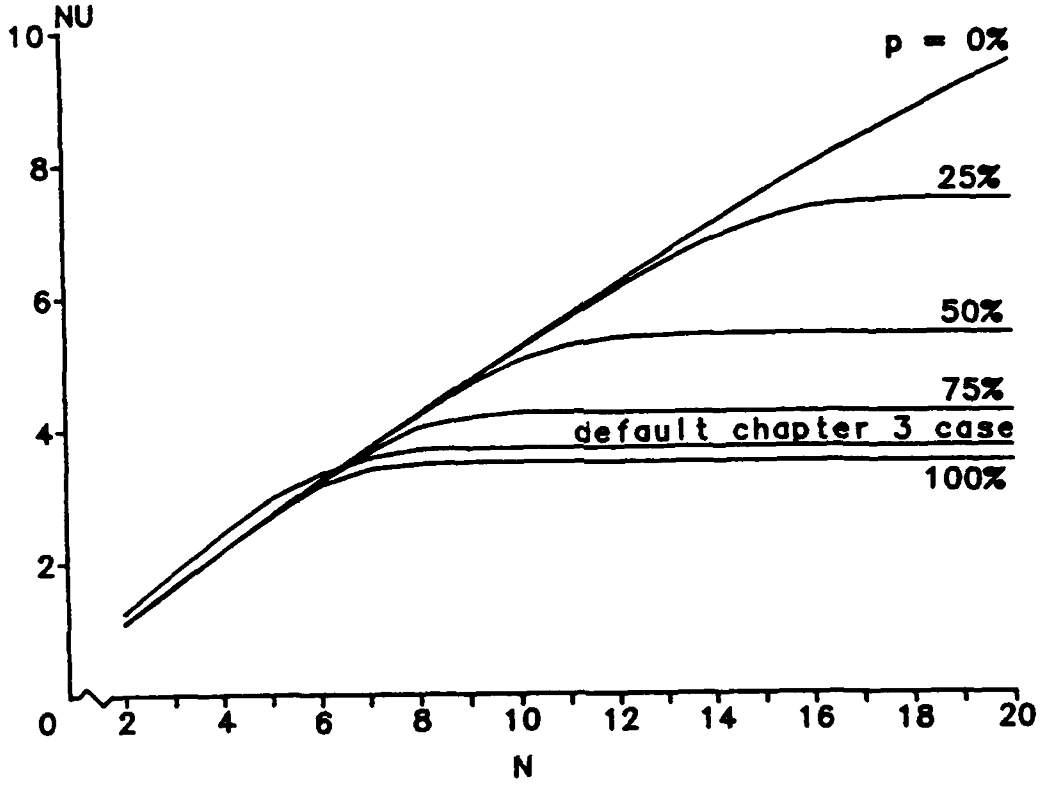


Figure 4-1. System performance vs. number of processors. Effect of presence percentage on system performance.

values of *p* will be much lower than 90%. Therefore, the subset requirement provides better system performance for a large number of processors. Directory information in L2 cache to reduce the bus usage is a good idea if it is desired to efficiently utilize a large number of processors.

CHAPTER 5

CONCLUSION

There has been a general trend over the years to build computer systems with faster processors and larger memories. This is the result of faster technologies and better architectures being used. There is no reason to believe that this trend will not continue in the future. As this trend continues, the difference in processor cycle times and memory access times increases. This difference is already large enough for cache memories to be advantageous. Most computer systems currently on the market contain cache memories. Faster processors will require faster caches to be efficient. These caches must be large enough to provide low miss ratios. It then becomes a trade-off whether to use larger and slower caches to provide low miss ratios or use smaller and faster caches to efficiently utilize faster processors. This thesis proposes that both be used: one level of smaller and faster cache memory and one level of larger and slower cache memory. The faster the processors become, the more the need for two-level caches. Therefore, the need for two-level caches will grow as the trend toward faster processors continues.

Another trend has been the increasing amount of logic that can be placed on a single chip. It is already possible to place an entire processor on a single chip. As this trend continues, it will become possible to place cache memory on the same chip with a processor. Since data can be transferred on a chip much faster than between chips, this cache memory will be very fast. It will also most likely be quite small, with a high miss ratio. Accessing main memory for each cache miss would not give very good performance. A second level of cache between the chip and main memory would probably provide much better performance.

The advantages of multiprocessors have already been discussed. These advantages should make them more popular in the future. This thesis has demonstrated the usefulness of two-level caches for multiprocessors. Most of the parameters used in the performance model were only estimated, but it has been shown that a two-level cache is effective for a multiprocessor over a large range of these parameters. More research, though, is needed in this area as many questions remain unanswered. The parameters can be better estimated by running program traces through a computer simulation of the proposed system. The effect of increasing the L2 cache miss ratio, the degree of sharing, and cache interference with an increasing number of processors needs to be studied. The optimum cache and block sizes also need to be determined. Future trends and research will most likely lead to the use of two-level caches in future computer systems.

REFERENCES

[Archibald84] J. Archibald and J. L. Baer, "An Economical Solution to the Cache Coherence Solution," *Proc. of 11th Int. Symp. on Computer Architecture*, June 1984, pp.355-362.

[Bean79] B. M. Bean, K. Langston, R. Partridge, and K. K. Sy, "Bias Filter Memory for Filtering out Unnecessary Interrogations of Cache Directories in a Multiprocessor System," United States Patent 4,142,234, February 27, 1979.

[Censier78] L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Trans. Comput.*, vol. C-27, December 1978, pp. 1112-1118.

[Conti69] C. J. Conti, "Concepts for Buffer Storage," *IEEE Comput. Group News*, vol. 2, March 1969, pp. 9-13.

[Goodman83] J. R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *Proc. 10th Int. Symp. on Computer Architecture*, June 1983, pp. 124-131.

[Kaplan73] K. R. Kaplan and R. O. Winder, "Cache-Based Computer Systems," *Computer*, March 1973, pp. 30-36.

[Meade70] R. M. Meade, "On Memory System Design," *AFIPS Proc. FJCC*, vol. 37, 1970, pp. 33-43.

[Papamarcos84] M. S. Papamarcos, "A Low Overhead Coherence Solution for Bus-Organized Multiprocessors with Private Cache Memories," CSG-29 Technical Report, Coordinated Science Laboratory, Urbana, Illinois, May 1984.

[Patel82] J. H. Patel, "Analysis of Multiprocessors with Private Cache Memories," *IEEE Trans. Comput.*, vol. C-31, April 1982, pp. 296-304.

[Rao78] G. S. Rao, "Performance Analysis of Cache Memories," *J. ACM*, vol. 25, No. 3, July 1978, pp. 378-395.

[Rudolph84] L. Rudolph and Z. Segall, "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors," *Proc. of 11th Int. Symp. on Computer Architecture*, June 1984, pp. 340-347.

[Smith82] A. J. Smith, "Cache Memories," *Computing Surveys*, vol. 14, No. 3, September 1982, pp. 473-530.

[Strecker76] W. D. Strecker, "Cache Memories for PDP-11 Family Computers". *Proc. 3rd Annual Symp. on Computer Architecture*, January 1976, pp. 155-158.

[Tang76] C. K. Tang, "Cache System Design in the Tightly Coupled Multiprocessor System," *AFIPS Proc. NCC*, vol. 45, 1976, pp. 749-753.

[Widdoes79] L. C. Widdoes, "S-1 Multiprocessor Architecture (MULT-2)," 1979 Annual Report - The S-1 Project, Volume 1: Architecture, Lawrence Livermore Laboratories, Technical Report UCID-18619, 1979.

[Yeh83] P. C. C. Yeh, J. H. Patel, and E. S. Davidson, "Shared Cache for Multiple-Stream Computer Systems," *IEEE Trans. Comput.*, vol. C-32, January 1983, pp. 38-47.

[Yen82] W. C. Yen and K. S. Fu, "Coherence Problem in a Multicache System," *Proc. 1982 Int. Conf. on Parallel Processing*, 1982, pp. 332-339.